SÓLO PROGRAMADORES

Revista especializada para usuarios de PC

AÑO 2. Nº 11
1250 PTAS.

ARGENTINA 9'50 $
CHILE 3000 $

Escrita por desarrolladores para desarrolladores

Cómo construir una demo

Aproximación a los grandes sistemas

Programas residentes bajo DOS

Watcom/386 C++

TORNEO DE PROGRAMACIÓN DE DEMOS

Y además:

- Tratamiento digital de la imagen
- Técnicas de sonido
- Curso de programación bajo Windows
- Formatos gráficos
- Programación C++
- Curso de ensamblador

9 778411 344792 00011

TOWER COMMUNICATIONS, S.A.L.

MASTER DE PROGRAMACION EN WINDOWS

Suscribase hoy mismo llamando al (91) 741 26 62

PROGRAMACION EN

WINDOWS

VISUAL BASIC
VISUAL C++

EL PLACER DE PROGRAMAR EN WINDOWS

SÓLO PROGRAMADORES
Revista práctica para usuarios de PC

CARTA DEL DIRECTOR

VOLVEMOS A LA CARGA

Bien es cierto que algunos lectores nos han comentado ultimamente eso de: "a mi el Unix, Linux y similares me gustan bastante pero *echo de menos* algo más práctico..." También son unos cuantos los admiradores de las *demos* del pasado Assembly, aquellas que dimos en el número 2 de SÓLO PROGRAMADORES. Y qué decir de las cartas que hemos recibido reclamando la continuidad del curso de programación de sonido, escrito por Luis Crespo, el autor del conocido Visual Player. Bueno amigos lectores, nos habéis convencido. Es hora de *poneros* a todos a programar. Y ya puestos, vamos a hacer algo que no sirva para nada en concreto. Nada de editores de sonido, nada de programas de gestión... Vamos a enseñaros a programar DEMOS, (y de las buenas).

A partir de este mes comenzamos una nueva serie de artículos encaminados a la programación de efectos y rutinas gráficas, similares a los que se incluyen en las demos de grupos de programación como Future Crew, Triton o Legend Design. Esperamos que esos mismos efectos puedan ser utilizados para adornar vuestros propios programas.

Aprovechando que viene el verano y hay mucho tiempo libre para programar, hemos decidido convovar un concurso de programación de DEMOS, al hilo de nuestra serie de artículos. Esperamos la participación de todos nuestros lectores. Para ello, nada mejor que un suculento premio de 250.000 ptas en efectivo al ganador de la primera categoría. Las bases y normas del concurso se publicarán en el número de AGOSTO de SÓLO PROGRAMADORES, así que os esperamos el próximo mes.

Este mes y tras un paréntesis continuamos la serie de artículos de sonido con el análisis de los conocidos ficheros MOD, procedentes del Amiga. En el apartado de artículos puntuales explicamos cómo hacer programas residentes desde ensamblador. La primera entrega de la serie de programación de demos y efectos gráficos está dedicada al efecto del fuego, seguramente ya conocido por muchos de nuestros lectores. Para finalizar, un breve análisis del compilador de moda en C++, el Watcom. De momento analizamos a vista de pájaro las cualidades del compilador elegido por la mayoría de los programadores de juegos. En próximos números profundizaremos en sus características.

Bueno, esto es todo por este mes. Nos despedimos no sin antes llamar la atención sobre una nueva publicación *pariente* de SÓLO PROGRAMADORES en la que estamos implicados algunos de los miembros de esta redacción, su nombre PC PLAYER y, como ya habréis adivinado está dedicada integramente al mundo de los video juegos para PC. Os la recomendamos.

Nos vemos el próximo mes con alguna que otra *sorpresa*. Hasta entonces, feliz verano.

MARIO DE LUIS

SÓLO PROGRAMADORES
Revista práctica para usuarios de PC

JULIO 1995. Número 11
Es una publicación de

TOWER COMMUNICATIONS, S.R.L.

Editor
Antonio M. Ferrer Abelló
Directora Comercial
Carmina Ferrer
Director de Producción
Carlos Peropadre
Publicidad
Magdalena Pedreño Llorente

••••••

Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador técnico
Emilio Postigo Riancho

Colaboradores
Juan Manuel Martín, Luis Martín, José María Peco, Enrique de Alarcón, Santos Riancho, José Carlos Remiro, Pedro Antón, Luis Crespo, Juan José Samper, José Galán, Jorge R. Regidor, Agustín Guillén, Enrique Coiras, David Aparicio, Fernando de la Villa, Ignacio Cea.

Edición técnica
Emilio Castellano
Maquetación
Fernando García Santamaría
Tratamiento de imagen
Josefa Fernández Martínez
Servicios Informáticos
Digital Dreams Multimedia, S.L.
Ilustraciones
Miguel Alcón
Secretaría de Redacción
Consuelo Jiménez
Suscripciones
Erika de la Riva
Redacción, Publicidad y Administración
C/ Marqués de Portugalete, 10
28027 MADRID
Telf.: 741 26 62 / Fax: 320 60 72
Filmación
Duvial
Impresión
G.D.B.
Distribución
MIDESA

La revista SÓLO PROGRAMADORES no tiene porqué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados. El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792

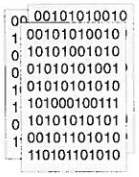
SUMARIO

6 **CONCURSO**
Sólo Programadores convoca el primer concurso de demos. Esta es la oportunidad de demostrar tus habilidades, ideas y dominio del PC.

8 **NOTICIAS**
Las novedades más interesantes del mundo informático:: compiladores, herramientas, lenguajes, nuevos sistemas..., y comentarios de libros de interés.

14 **ENTREVISTA**
Antonio Linares, autor de las librerías FIVEWIN, el producto software español de mayor difusión, nos habla del origen y desarrollo de éstas.

17 **CURSO DE ENSAMBLADOR**
En esta nueva entrega del curso de ensamblador se toma contacto con las directivas de control simplificado de segmento, que permiten establecer, de forma rápida y completa, el esqueleto de un programa ensamblador.



23 **FORMATOS GRÁFICOS**
En este último capítulo de la serie se estudia la revisión 5 del formato TIFF, efectuando las simplificaciones necesarias para abordarlo con claridad.

27 **CURSO DE PROGRAMACIÓN BÁSICA**
Cuando se desarrolla un programa es necesario tener las ideas claras. Este capítulo muestra la forma de abordar un problema genérico de manera que las dificultades del proceso de programación queden reducidas al mínimo.

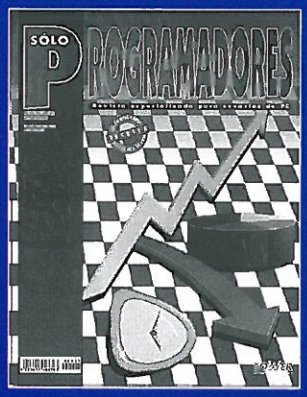
32 **HERRAMIENTAS GRÁFICAS**
Seguimos extrayendo potencia de nuestro equipo. En esta tercera entrega se introducen métodos para el manejo de imágenes en color y la técnica del entramado.

35 **CÓMO HACER UNA DEMO**
Comienzo de un conjunto de artículos que mostrarán al lector cómo construir paso a paso una demo, explicándose por separado las maneras de obtener los distintos efectos que forman parte de ellas.

38 **TÉCNICAS DE SONIDO**
Reanudación de la serie dedicada a este tema. En este artículo se explica cómo lograr que un programa haga sonar un fichero MOD, empleando en su desarrollo rutinas en Pascal y ensamblador.

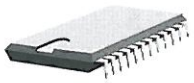
43 **SISTEMAS EXPERTOS**
Este capítulo aborda el problema de la adquisición de conocimientos por parte de un sistema experto e introduce conceptos fundamentales para su comprensión.

Sólo Programadores saldrá también a la calle en el mes de Agosto, esta vez acompañado de un interesante libro sobre el compilador Watcom C++. También se publicarán las bases completas del Primer Concurso Internacional de Programación de Demos. Si quieres conocer la forma más divertida de ganar 250.000 Ptas. en efectivo, no faltes a tu cita veraniega. Te esperamos.

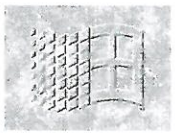


SUMARIO

49 **PROGRAMAS RESIDENTES BAJO D.O.S.**
Con este artículo se inicia una serie que permitirá al lector programar sus propios TSRs. Se hará hincapié en los aspectos esencialmente problemáticos.



53 **CURSO DE PROGRAMACIÓN BAJO WINDOWS**
En este capítulo del curso se inicia la construcción paso a paso de un programa editor. La potencia del control EDIT será empleada para este fin.

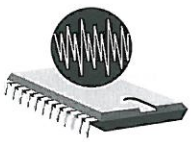


58 **SISTEMA OPERATIVO LINUX**
Linux posee una potente librería gráfica que permite explotar nuestra tarjeta SuperVGA con la comodidad propia de otros entornos. En este artículo se exploran sus posibilidades.

62 **PROGRAMACIÓN EN C++**
Con este artículo comienza una serie que desvelará los algoritmos y técnicas empleados en la construcción de un sistema de ventanas.

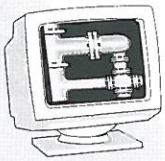
67 **GRANDES SISTEMAS**
Esta nueva sección de la revista va dirigida a aquellas personas que, encargadas del desarrollo de aplicaciones en sistemas de alto nivel, pueden necesitar en un momento determinado explotar su entorno al 100%.

71 **EL TEMPORIZADOR 8253/8254**
La posibilidad de controlar intervalos de tiempo reales independientemente del sistema en que corre una aplicación es el tema de este artículo.



74 **PROGRAMACIÓN EN CLIPPER**
En este artículo se estudian los niveles reader y de funciones para la edición de variables y campos, empleando la programación orientada a objetos. Un paso más en el estudio de este potente lenguaje.

79 **HERRAMIENTAS DE PROGRAMACIÓN**
Análisis de las características y prestaciones del paquete compilador multisistema Watcom/386 C++, el más completo del mercado: IDE, ejecutables para varios entornos, diversos editores y compiladores, etc.



82 **CORREO DEL LECTOR**
Esta sección se encarga de solucionar las dudas y problemas encontrados por nuestros lectores en cualquier ámbito de la programación.

RECTIFICACIÓN: El autor del artículo "FIVEWIN, Clipper bajo Windows", publicado en el nº 10 de esta revista es Ignacio Ortiz y no Enrique Velasco. Sólo Programadores pide disculpas por las molestias causadas.

1er Torneo de Demos e infografía

La revista Sólo Programadores organiza un nuevo torneo. Tras el éxito conseguido con el concurso de las 4 en raya, pensamos ahora en continuar fomentando el espíritu de la programación y la creatividad con un reto muy distinto.

En la anterior iniciativa apostamos por la lucha de programas inteligentes, que incorporaban complicados algoritmos para conseguir hacer las 4 en raya en el menor número de movimientos y siempre contra reloj. La sorpresa fue grata al ver elevado número de participantes y sobre todo lo bien que estaban hechos muchos de esos programas.

Esta vez la cosa es muy distinta. El nuevo torneo está pensado para aquellos que cuentan con grandes dosis de creatividad y diseño.

Es una llamada que pondrá a prueba a los programadores de demos y a aquellos que no programan pero tienen cualidades gráficas.

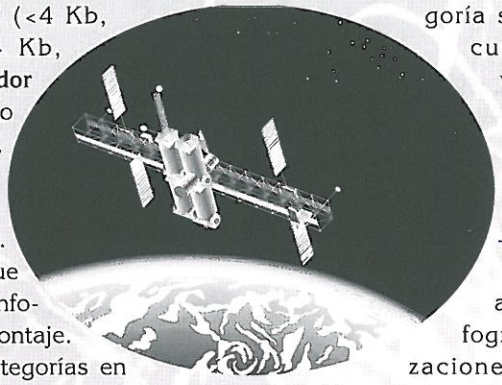
Todos los años se organizan distintos campeonatos de programación de demos e infografía en distintos lugares del planeta, y lo que está demostrado es que cada vez cuentan con más seguidores.

Como revista especializada era obligado hacer un torneo de **demos e infografía** porque creemos que en nuestro país hay cada vez más programadores con ganas de demostrar su capacidad para asombrarnos con sus trabajos.

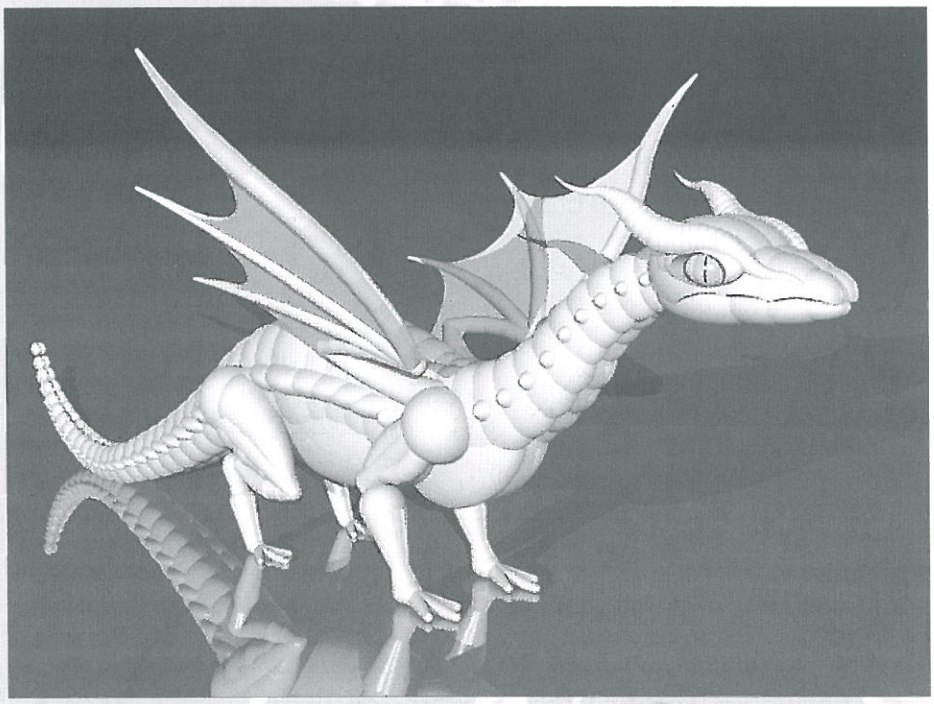
Las puertas estarán abiertas en los próximos meses para aquellos que quieran ponerse a prueba consigo mismos enviándonos una **demo, infografía, digitalización** o **morphing** que nos deje "a colores" (nunca mejor dicho).

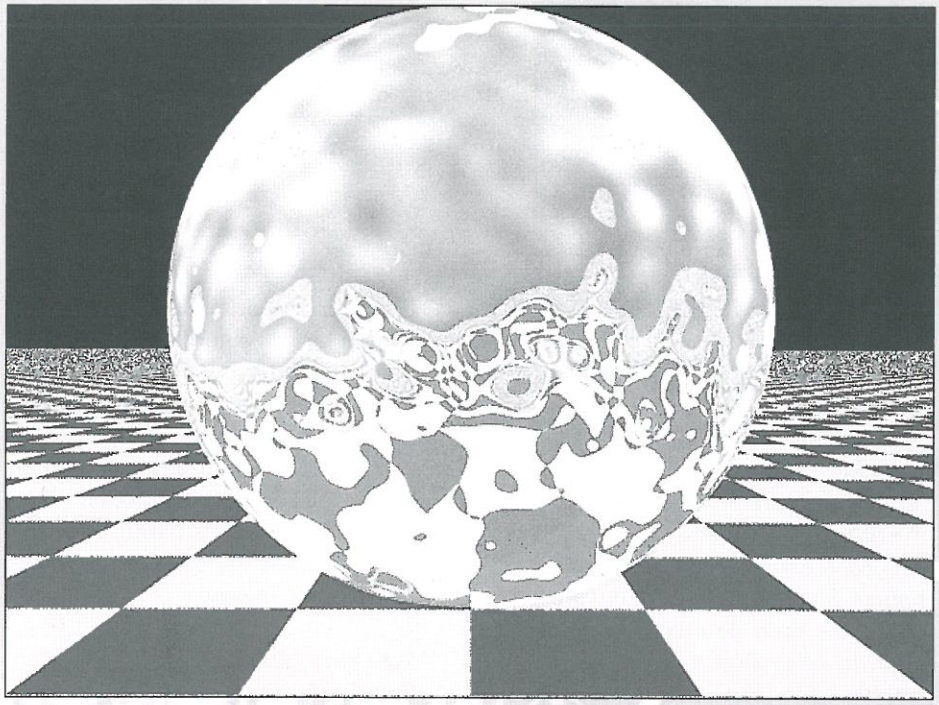
La competición tendrá diversas categorías y subcategorías.
La estructura será la siguiente:

- **Demos.** Se englobarán en distintas categorías atendiendo principalmente a su **longitud** (<4 Kb, <64 Kb, <1024 Kb, etc.) y al **procesador** mínimo requerido (386, 486, Pentium, etc.).
- **Infografía.**
  - **Imagen fija.** Imágenes fijas que representen una infografía o un fotomontaje. Aquí se harán categorías en función de la resolución y el número de colores de cada imagen.
  - **Animaciones.** Aquí se encontrarán los trabajos realizados con programas de render (3D Studio, POV, Crystal Topas, etc.).
- **Digitalización.** Este apartado está pensado para aquellos que tengan digitalizaciones de vídeo en formato AVI, MPEG, etc. Para esta categoría se podrá enviar cualquier tipo de vídeos caseros siempre y cuando sean de elaboración propia.
- **Morphing.** Se admitirán las animaciones (infografías o digitalizaciones) que utilicen técnicas de transformación de imágenes.



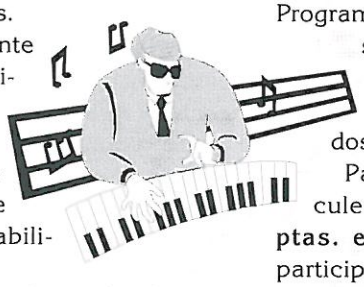
A la hora de hacer la competición se tendrán en cuenta diversas caracte-





rísticas en cada categoría. En las demos, la calidad de los efectos visuales será la nota predominante en la valoración total, puesto que es en este aspecto donde se ve la capacidad de programación y creatividad de los grupos.

El sonido es la otra cualidad significativa de una demo. Aquí es tan importante la composición de la melodía (Luis Cobos enviadiaría a más de un compositor de MODs) como la programación de las rutinas de la música y los efectos sonoros.



Cada participante o grupo de participantes podrá enviar uno o varios trabajos en los que se pongan de manifiesto sus habilidades.

Los trabajos serán evaluados por un jurado compuesto por miembros de la redacción de Sólo Programadores e integrantes de diversos grupos de programación, que valorarán distintos aspectos de los trabajos enviados.

Para el ganador habrá una suculenta recompensa de **250.000 ptas. en metálico.** Para los demás participantes que hayan quedado en segundos y terceros puestos habrá **numerosos regalos.**

Las bases del concurso se podrán conseguir en el ejemplar de agosto de Sólo Programadores. En dicho ejemplar también se podrá obtener la tarjeta de participación imprescindible para concursar. La fecha inicial para la recepción de los trabajos es el 15 de agosto.

IMPORTANTE

NO SE ADMITIRÁN TRABAJOS ANTES DEL 15 DE AGOSTO. PARA CONCURSAR ES IMPRESCINDIBLE ENVIAR LA FICHA DE PARTICIPANTE QUE SE ENCONTRARÁ DISPONIBLE EN EL NÚMERO DE AGOSTO DE SÓLO PROGRAMADORES.

250.000 ptas. en metálico y cientos de programas para los ganadores

(no se admitirán trabajos hasta el 15 de agosto)

BUSCAMOS A LOS MEJORES

- Programadores en C/C++
- Grafistas, diseñadores, dibujantes
- Expertos en lenguaje ensamblador
- Programadores 3D
- Expertos en Sonido
- Músicos con nociones de MOD, MIDI
- Programadores de juegos
- Animadores gráficos
- Infografistas con experiencia en 3D Studio, Photoshop, Deluxe Paint Animation
- Programadores con dominio de lenguajes multimedia: Authorware, Toolbook, Visual Basic, Macromedia Director, etc
- Expertos en comunicaciones

PARA DESARROLLAR

SOFTWARE MULTIMEDIA:

Presentaciones, libros interactivos, programas educativos, sistemas de comunicaciones, centros servidores de datos, videojuegos y mucho más...

SI ERES programador, músico o infografista y tienes ideas o proyectos en estudio de desarrollo ponte ya en contacto con nosotros.

TE OFRECEMOS las mejores condiciones y apoyo para producir tus programas y venderlos en el mercado nacional y extranjero. Formación en nuevas tecnologías y la mejor biblioteca de rutinas gráficas y sonido para desarrolladores.

Aportamos gráficos, rutinas o música, según las necesidades, para complementar cada proyecto.

Si estás interesado en unirte a una de las empresas más punteras en alta tecnología y desarrollo de software, no dejes pasar esta oportunidad. Envíanos carta con tus datos personales (curriculum vitae, con una muestra de tus anteriores trabajos) y un teléfono de contacto a:

DDM DIGITAL DREAMS MULTIMEDIA

DIGITAL DREAMS MULTIMEDIA
Ref. Programadores
C/ Vicente Muzas 15, 1° D - 28043 MADRID
Tf.: (91) 519.23.53 Fax (91) 413.55.77
BBS: (91) 519.75.75 Internet: ddm@servicom.es

SILICON GRAPHICS CREA UNA FILIAL

Silicon Studio es el nombre que recibe la nueva compañía subsidiaria de Silicon Graphics. Está dedicada a la creación de herramientas de desarrollo y a proporcionar soporte para los mercados multimedia. Realizará acuerdos estratégicos con otras compañías para desarrollar software para diferentes tipos de mercados, como el cine, la televisión interactiva, videojuegos, editoriales, vídeo y otros.

El concepto de Silicon Studio fue presentado por Silicon Graphics en el año 1993, cuando introdujo una serie de nuevos productos y aplicaciones de software de terceros para la producción de películas y videos digitales.

Algunos de los proyectos actuales de Silicon Graphics incluyen la asociación con Time Warner Cable en una prueba de Red de Servicios Digitales Interactivos, y el acuerdo con Nintendo para el desarrollo conjunto de un sistema de videojuegos.

Para más información:
Silicon Graphics, S.A.
Plaza del descubridor Diego de Ordás, 3
Edificio "Santa Engracia 120"
28003 Madrid
Tel: (34-1) 442 90 77
Fax: (34-1) 442 01 50

WESTERN DIGITAL PRESENTA LA PRIMERA UNIDAD ENHANCED IDE DE 3,5 PULGADAS Y 1,6 GB DE CAPACIDAD

La nueva unidad Caviar AC31600 está diseñada para proporcionar alta capacidad y rendimiento en los PC basados en el procesador Pentium. Se compone de tres discos de 3,5 pulgadas de 540 Mb, formando un total de 1,6 Gb de capacidad.

Proporciona un tiempo medio de búsqueda en las lecturas por debajo de 10 ms, una latencia de eje giratorio de 5,76 ms y una velocidad de rotación de 5200 revoluciones por minuto. Ofrece compatibilidad garantizada y un tiempo medio de fallo de 300.000 horas.

La unidad Caviar AC31600 se suministra ahora en los nuevos sistemas AT&T, Dell y Gateway.

Para más información:
Paul Calderwood
Western Digital Europe
45 Pampisford Road
Purley
Surrey CR8 2NJ
Reino Unido
Tel: +44 181 763 2241

NUEVOS HUBS DE COMBINACIÓN DEPARTAMENTAL LANPLEX PARA REDES LOCALES DE ALTO RENDIMIENTO

La compañía Anixter ha anunciado el comienzo de la distribución de la familia LANplex 2000 de Hubs de conmutación Ethernet y Ethernet/FDDI inteligentes y de alto rendimiento 3Com.

Los conmutadores LANplex incluyen soporte a redes virtuales, expansión en árbol, filtrado de paquetes, y elección entre puente 802.1d y fragmentación IP. Su motor de conmutación proporciona rendimiento para más de 565.000 paquetes por segundo, y emplea una técnica de almacenamiento elástico de paquetes en buffer.

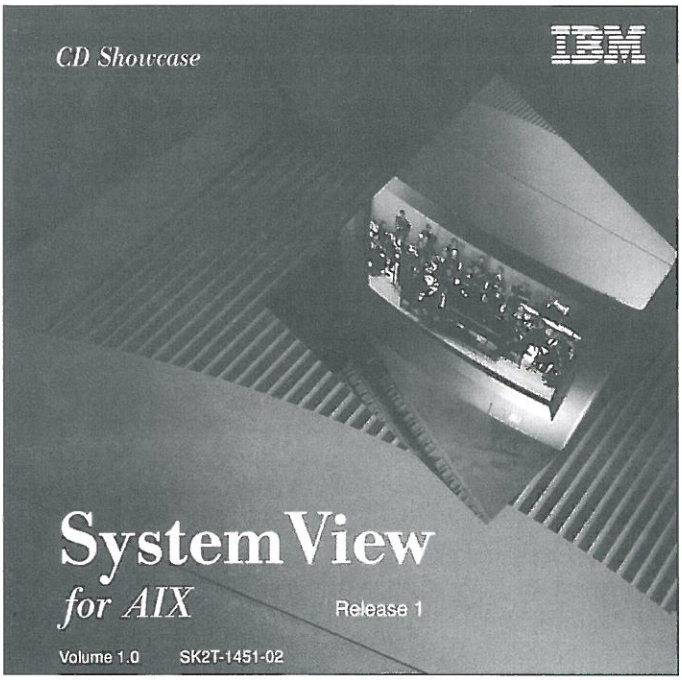
El conmutador LANplex 2500 ofrece hasta 16 conexiones Ethernet conmutadas a 10 Mbps y dos puertos de alta velocidad conmutados. El LANplex 2016 proporciona hasta 16 conexiones Ethernet conmutadas.

Para más información:
Anixter España S.A.
Progreso, 65-67

Polígono industrial Almeda
08940 Cornellá Llob., Barcelona
Tel: (93) 474 06 61
Fax: (93) 474 08 76

NUEVA VERSIÓN DE MICROSOFT SNA SERVER

Microsoft ha presentado la versión 2.11 de Microsoft SNA Server que se ejecuta sobre el sistema operativo Microsoft Windows NT Server. Incorpora soporte a los nuevos sistemas PowerPC y proporciona nuevas características que aumentan la integración del SNA Server con aplicaciones y datos de host IBM.



Entre las nuevas características destacan: soporte para aplicaciones SNA de 16 bits sobre Windows NT, soporte para clientes de TN3270, servicio de acceso remoto SNA, soporte al API EHNAPPC, soporte para el nuevo Client Access/400 de IBM, utilidad AFTP y ODBC/DRDA. SNA Server 2.11 estará disponible a partir del día 12 de junio del presente año.

Contacto de prensa:
José González Rivera / Mario Gutiérrez
Spanish Executive Information Service (S.E.I.S.)
Tel: (91) 308 43 15
Fax: (91) 310 38 76

INTERVENIDA POR PRESUNTA PIRATERÍA INFORMÁTICA LA ORGANIZACIÓN NACIONAL DE CIEGOS DE ESPAÑA (ONCE)

La Guardia Civil de Badajoz efectuó el pasado 9 de mayo un exhaustivo registro por presunto delito informático en la Delegación Territorial de la O.N.C.E. en Badajoz, siguiendo instrucciones del Juzgado de instrucción nº 1 de la misma ciudad.

La Comisión Judicial investigó los programas instalados en los ordenadores de la delegación de la O.N.C.E. mediante un sofisticado programa diseñado para tal fin. Según el informe, existen fundadas sospechas de uso no autorizado de programas en las instalaciones de la O.N.C.E.

Es el primer registro procedente del Programa de Recompensas Económicas de Business Software Alliance (BSA), finalizado el 15 de abril y mediante el cual se han obtenido 344 denuncias en firme hasta la fecha. BSA ha presentado ya otros 11 casos en varios juzgados españoles.

Para más información:
Burson-Marsteller
Javier Celaya / Ana Isabel Alcaide
Tel: (91) 555 32 13

INTERGRAPH FIRMA UN ACUERDO CON MICROSOFT

Microsoft Ibérica ha anunciado el acuerdo alcanzado con Intergraph bajo el programa Solution Provider. Mediante este acuerdo, la empresa Intergraph implementará los productos de Microsoft en sus servicios dirigidos a completar su gama de productos. Intergraph Corporation se dedica al desarrollo de sistemas gráficos informáticos para la gestión corporativa y sistemas CAD/CAM/CAE.

El programa Solution Provider de Microsoft ofrece a sus clientes servicios de empresas especializadas con la total garantía de Microsoft Ibérica. Solution Provider ha conseguido ya más de 100 acuerdos en nuestro país.

Contacto de prensa:
José González Rivera / Elena Arroyo
Spanish Executive Information Service (S.E.I.S.)
Tel: (91) 308 43 15
Fax: (91) 310 38 76

ORACLE PRESENTA ORACLE7 MULTIDIMENSION PARA EL MERCADO GIS

Oracle ha presentado Oracle7 Multidimension, una nueva tecnología que posibilita la gestión de información espacial y de información atribuida (tabular tradicional) desde una única base de datos.

Con Oracle7 Multidimension desaparece la necesidad de utilizar un sistema propietario de base de datos espacial. Permite la construcción de sistemas de apoyo a la toma de decisión sobre una arquitectura de base de datos común capaz de gestionar todo tipo de información: imágenes, sonido, texto e información espacial. Incluye extensiones SQL y PL/SQL y generación de un único método de partición de datos, entre otras características.

Oracle7 Multidimension es una extensión del servidor de bases de datos Oracle7 y a partir de mayo estará disponible para Solaris, OSF/1 y HP-UX.

Para más información:
Oracle Ibérica
Cristóbal Orenes
Tel: (91) 335 91 00

SYBASE Y COMPAQ FIRMAN UN ACUERDO PARA DESARROLLAR SOLUCIONES CLIENTE/SERVIDOR

Sybase Inc. y Compaq Computer Corporation trabajaran conjuntamente en actividades de marketing y en el desarrollo de soluciones cliente/servidor optimizadas gracias a la firma de un acuerdo a nivel mundial.

El acuerdo tendrá como consecuencia el establecimiento de Compaq como plataforma de futuros desarrollos de Sybase bajo Windows NT, SCO Unix y Novell UnixWare. También se realizarán test de rendimiento TPC-C, pruebas de clientes y tests de utilidad, integración y certificación; formación de personal de ventas y desarrollo de ambas compañías, colaboración con Vendedores de Software Independientes (ISVs) y resolución de problemas técnicos entre el software de Sybase y los sistemas Compaq.

Contacto de prensa:
Marcos Carmena / Ludi García
Spanish Executive Service Information (S.E.I.S.)
Tel: (91) 308 43 15
Fax: (91) 410 38 76

IBM ANUNCIA UNA NUEVA GENERACIÓN DE SOFTWARE PARA GESTION DE SISTEMAS

IBM ha anunciado las series SystemView, que responden a las necesidades de los usuarios de entornos cliente/servidor. Estas series simplifican la gestión de sistemas distribuidos compuestos por equipos de distintos fabricantes.

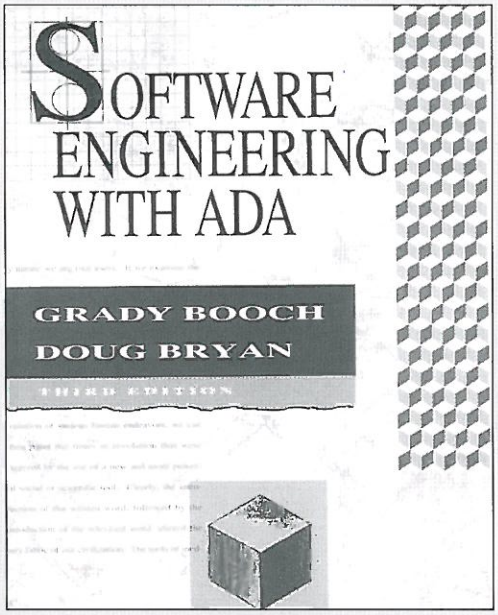
SystemView proporciona un conjunto integrado de aplicaciones y servicios orientados a procesos, dirigidos mediante un interface de usuario fácil de utilizar. La orientación a procesos, a diferencia de la orientación a productos, permite gestionar un sistema de manera más sencilla y eficaz. Incluye una capacidad de análisis proactivo capaz de resolver problemas antes de que el usuario llegue a notarlo.

Por ahora sólo está disponible SystemView para AIX. Sin embargo, IBM ampliará el desarrollo a otras plataformas como OS/2 y MVS, previstas para finales de año. Las soluciones SystemView para OS/400 tienen prevista su comercialización a principios de 1996.

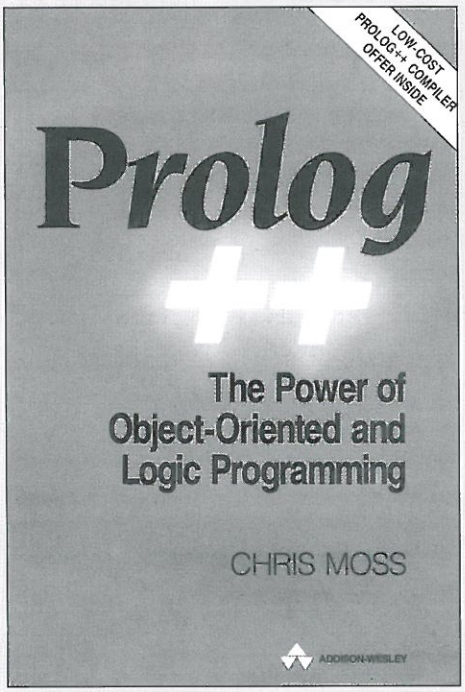
LIBROS

SOFTWARE ENGINEERING WITH ADA (THIRD EDITION)

Este libro constituye una completa referencia de ADA dirigida tanto al principiante como al analista que desean aprender las aplicaciones de este potente lenguaje. En él se tratan a fondo las características de ADA, con numerosos ejemplos que ilustran el modo en el que se diseña un programa con este lenguaje. El objetivo de la obra no es sólo exponer las características del lenguaje, sino mostrar cómo se debe aplicar el lenguaje en la creación de grandes sistemas de software. Para ello se expone un método de desarrollo orientado a objetos que aprovecha la potencia de ADA para la implementación de grandes sistemas.



Editorial:
The Benjamin/Cummings Publishing Co., Inc.
Autores: Grady Booch y Doug Bryan
575 páginas
Idioma: Inglés
Precio: Consultar a la editorial (07-020-671 72 96)



PROLOG++. THE POWER OF OBJECT-ORIENTED AND LOGIC PROGRAMMING

Prolog++ es un nuevo lenguaje de programación ideado para combinar las mejores características de la programación orientada a objetos y de la programación lógica. Para entender la obra no son necesarios conocimientos previos en ninguno de los dos tipos de programación. El libro comienza con una introducción a la programación orientada a objetos y a Prolog, para entrar de lleno en la programación con Prolog++. Se dedica un capítulo a los interfaces gráficos de usuario, otro al diseño de programas y el último capítulo se centra en los objetos concurrentes y las bases de datos orientadas a objetos. Incluye multitud de ejemplos prácticos y una oferta para la adquisición de un compilador Prolog++ de bajo coste.
Editorial: Addison-Wesley
Autor: Chris Moss
326 páginas
Idioma: Inglés
Precio: Consultar a la editorial (07-020-671 72 96)

ESPERANDO EL MILAGRO ESPAÑOL

La industria del software de entretenimiento ha sufrido una transformación asombrosa en los últimos doce años. Cuando el ZX Spectrum y otras pequeñas maravillas de 8 bits estaban empezando a hacer furor entre un puñado de locos peligrosos, era relativamente fácil escribir un juego de calidad comercial. Sólo hacía falta encerrarse en una habitación con un manual de Ensamblador del Z80, algo de buen gusto para los gráficos y mucha ilusión. En aquel tiempo, llegó a florecer una diminuta pero prometedora industria en nuestro país. La formaban chavales llenos de ganas y de visión de negocio. No llegó a ser una edad de oro, pero de esta época salieron pequeñas joyas que todos recordamos con cariño.

En algún momento, algo fue mal. En la época en que los entrañables 8 bits fueron dando paso gradualmente a las nuevas y potentes máquinas, muchos se quedaron en el camino. Algunos se aferraron a los viejos cacharros y se hundieron con honor, como el capitán que se niega a abandonar el barco tragado por la tormenta. Otros, sencillamente, carecieron de los recursos necesarios para diversificarse entre las opciones del momento, y eligieron mal. Después de todo, cuando las posibles plataformas de destino eran el Commodore Amiga, el Atari ST y aquellos pecés caros y negados para los gráficos, ¿quién podía creer que la arquitectura de estos últimos sería la única superviviente? Yo no habría dado ni un duro por esos trastos de oficina. A otros, el error les costó el negocio.

Hoy las reglas del juego han cambiado para aquéllos que parten de cero. Las máquinas de hoy en día y los gustos del mercado se han aliado para minimizar las oportunidades de los desarrolladores modestos.

La gente quiere superproducciones, obras de software faraónico que ocupen varios CD-ROMs con horas de vídeo digital protagonizado por estrellas de Hollywood y rutinas tridimensionales capaces de quemarle a un Pentium la junta de la culata. Quieren centenas de fondos gráficos dibujados por artistas de caché y animaciones salidas de la trastienda de Disney. Así no se puede, hombre.

La situación me recuerda a la de la industria del cine. Tenemos un público que busca mayoritariamente la espectacularidad de gran presupuesto y reduce las posibilidades del realizador joven y falto de medios a intentar atraer la atención de una reducida minoría de divinos intelectualoides. Supongo que tendremos que estrujarnos el seso y echar el anzuelo en el río de las ideas para ver si se nos ocurre algo tan sencillito y revolucionario como el Tetris y demostramos a los juegoadictos que los placeres sencillos también pueden ser alucinantes.

¿Pero quién se va a comprar en las tiendas un jueguecito que sólo ocupe un disquete? Como no recibamos una subvención del Ministerio de Cultura, la inmensa mayoría de los desarrolladores españoles tenemos el acceso vedado a nuestras propias tiendas.

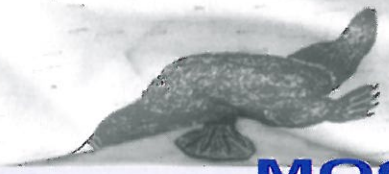
Pero no todo está perdido. Puede que de momento nos hayan expulsado de ese territorio, pero los mercados emergentes de los quioscos y el shareware tienen aún un sitio para los productos pequeños, siempre que haya calidad e imaginación. Un pequeño grupo de compañías españolas empieza a abrirse paso en el desolado campo de batalla. Los quioscos son un buen terreno para la compra por impulso de productos baratos, aunque se necesita un volumen de ventas muy alto para compensar el coste de distribución. El shareware empieza a ser un mercado "de verdad" en el extranjero, pero aquí en la piel de toro todavía no hay una cultura de apoyo a esta forma de distribución. Si conocen a alguien que haya pagado por un programa shareware en España, por favor tráiganlo aquí para que le hagamos un mapa del genoma y podamos aislar el pedacito de ADN responsable de tal comportamiento. Podríamos insertar el gen de la responsabilidad en un virus vector y diseminarlo por las páginas de la revista, a ver si infectamos a la población.

Creo que en el futuro próximo hay otra esperanza para los pequeños desarrolladores: las redes mundiales de comunicación como Internet (me resisto a emplear esa horterada de "las superautopistas de la información"). Tan pronto como se generalice el acceso a bajo coste de estas redes, y se habilite un sistema de pago electrónico popular y fiable, las pequeñas compañías podrán llegar a millones de usuarios en todo el globo con un gasto de distribución mínimo. Quizá en ese momento podamos jugar en un campo menos desigual; puede que ése sea el momento de levantar cabeza y asombrar al mundo. ■

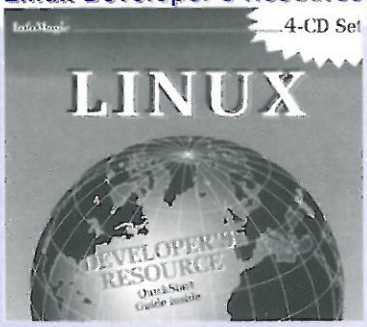
Linux:

Guerrilla UNIX Development

Venimus, Vidimus, Dolavimus



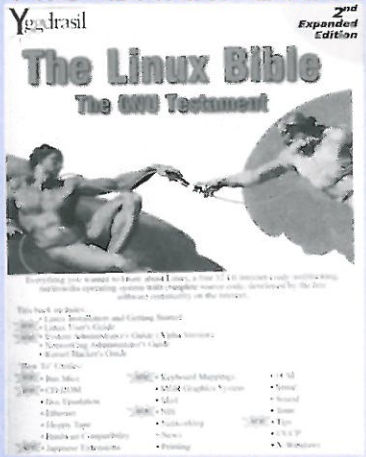
Linux Developer's Resource



El título más popular y actualizado sobre el sistema operativo Linux distribución Slackware 2.2 con kernel 1.2, y las recopilaciones más completas y recientes de los sistemas Internet (se lo mit.edu y prep.ai.mit.edu), una pequeña pero fundamental Guía de instalación y Primeros Pasos impresa, completa documentación on-line y extensa selección de paquetes, utilidades, programas y demos comerciales.

¡4 CDs! Precio de venta: **3.448** pts Precio 10+*: 2.450 pts

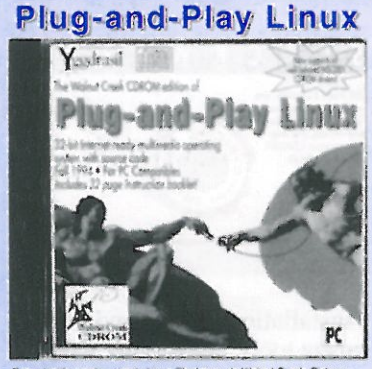
The Linux Bible



¡Casi 1200 páginas! Todo lo que siempre has deseado saber sobre Linux, el más potente sistema operativo de 32 bits de libre distribución, lo podrás encontrar aquí. Incluye guías de instalación y puesta en marcha, guías del usuario, guía del administrador del sistema, guía del kernel, configuración de la red, X-Windows y acceso Internet... ¡todo!

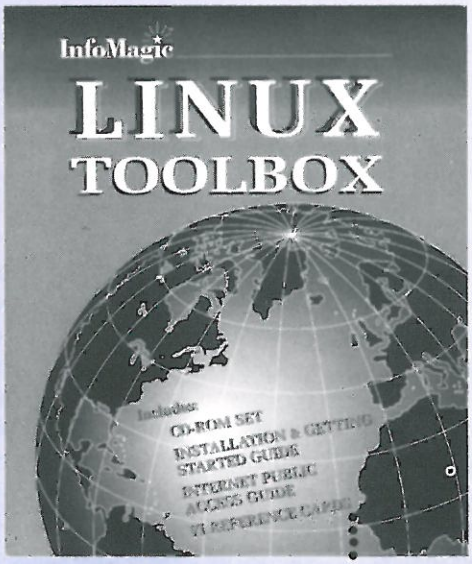
¡Nueva edición! Precio de venta: **6.466** pts Precio 10+*: 5.800 pts

Plug-and-Play Linux



Recopilación y selección de Linux Slackware de Walnut Creek. Sistema completo y código fuente, más una selección de los mejores archivos de sistemas Internet.

¡Novedad! Precio de venta: **3.017** pts Precio 10+*: 2.095 pts



Linux ToolBox

Paquete completo de utilidad y documentación para Linux. Incluye:

- Linux Developer's Resource (4 CDs).
- Libro Installation & Getting Started.
- Guía de acceso y uso de Internet.
- Tarjetas de referencia rápida del sistema operativo y utilidades.

¡Novedad! ¡4 CDs+libro! Precio de venta: **6.897** pts Precio 10+*: 5.800 pts

Plug-and-Play Linux

Distribución Linux de Yggdrasil, preconfigurada y lista para ser ejecutada directamente desde el CD. El mejor y más estable código de Linux, recopilado, configurado y ejecutado sin ocupar cientos de megas en disco duro.

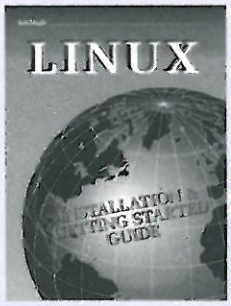
¡Novedad! ¡CD + Guía! Precio de venta: consultar

Linux Journal

¡La única revista mensual especializada en Linux. La fuente definitiva para conocer los últimos y mejores desarrollos, así como para obtener artículos serios e informativos, destinados tanto a usuarios como desarrolladores, del mundo fascinante y en constante cambio de Linux.

¡Novedad! Precio de venta: **1.724** pts

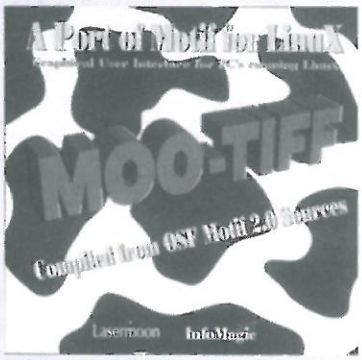
Linux Installation & Getting Started Guide



Libro de más de 200 páginas escrito por Matt Welsh, coordinador del proyecto de documentación de Linux. De utilidad tanto para principiantes como para usuarios avanzados, cubre los aspectos de la instalación, obtención del sistema, tutorial de uso, administración y configuración avanzada, así como otros muchos.

Precio de venta: **4.310** pts Precio 10+*: 3.900 pts

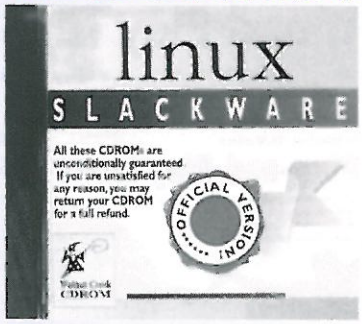
MOO-TIFF



¡Al fin! Una versión asequible del interface gráfico OSF/MOTIF 2.0 para PC's corriendo Linux. Runtime y entorno de desarrollo completo.

¡Novedad! Precio de venta: **17.241** pts Precio 10+*: 13.793 pts

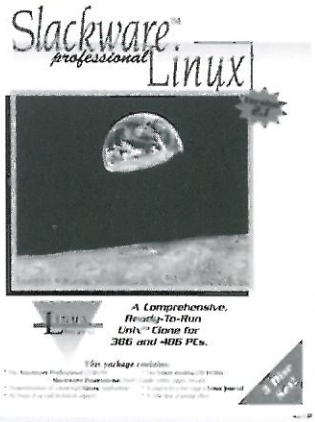
Linux Slackware



Recopilación y selección de Linux Slackware de Walnut creek. Sistema completo y código fuente, más una selección de los mejores archivos de sistema Internet.

¡Novedad! ¡2 CDs! Precio de venta: **3.017** pts Precio 10+*: 2.095 pts

Slackware Professional Linux



Paquete profesional y oficial de Linux, editado por Morse Telecommunication y con la supervisión del autor de la distribución Slackware, Patrick Volkerding. Incluye 3 CDs y libro de más de 500 páginas, cupones, ofertas, etc.

¡3 CDs + libro! Precio de venta: **8.190** pts Precio 10+*: 6.000 pts

Productos importados y distribuidos por

abc analog, s. l. *Directo*

(91) 6342000 / 6343213

(91) 6344786 / 6341293

Apartado de correos 157

28220 Majadahonda (Madrid)

Módem: (91) 6345100 / RDSI (91) 6347020

IVA no incluido en los precios

Precios válidos salvo error tipográfico y sujetos a cambio sin previo aviso

Todas las firmas y títulos mencionados están registrados por sus respectivos propietarios

*Consultar condiciones

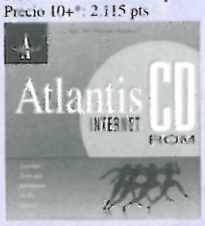
C/ Avrigenute 2, Esc. D, bajos A, B y C
28220 Majadahonda (Madrid)

RINCON PROFESIONAL

Más de 1.1 GB de documentación, información y código fuente para el lenguaje Ada. Extraído del sistema Internet PAT "Public Ada Library".

¡2CDs!

Precio de venta: **3.017** pts
Precio 10+*: 2.715 pts

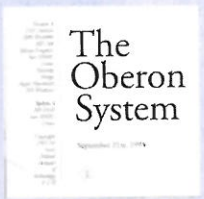


Colección de software de comunicaciones, programas de demostración, cuotas baratas en redes, utilidades y código fuente sobre Internet. Para entornos DOS, Windows, Mac y UNIX.

Precio de venta: **2.500** pts
Precio 10+*: 1.900 pts

Ficheros completos GNU, precompilados para Sun os 4.1.3 y Solaris 2.3.

Precio de venta: **3.879** pts
Precio 10+*: 2.995 pts



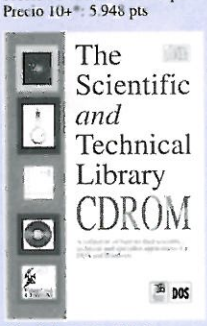
Archivos del lenguaje de programación y entorno operativo Oberon, extraídos del servidor ETH de Zurich.

Precio de venta: **2.586** pts
Precio 10+*: 2.155 pts

Prime Time TEXcetera

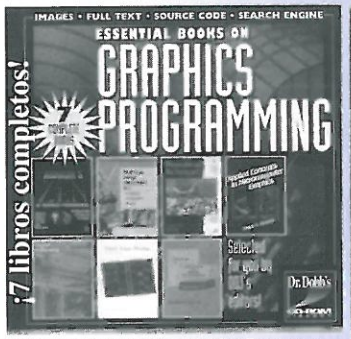
Colección de herramientas, fuentes, macros y documentación relativa a TEX. Extraído de los archivos CTAN. Incluye libro de 96 páginas.

Precio de venta: **6.897** pts
Precio 10+*: 5.948 pts



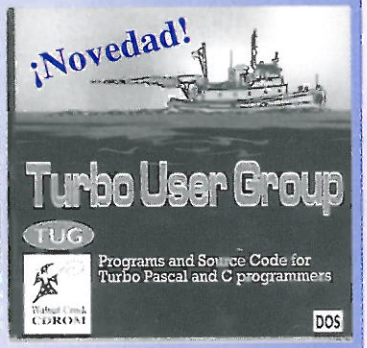
Colección de programas, utilidades y documentación científica y técnica para DOS y Windows. Recopilado por FDSL. Incluye libro de 96 páginas.

Precio de venta: **3.448** pts
Precio 10+*: 2.900 pts



7 libros profesionales seleccionados con la garantía de Dr. Dobb's: Algorithms for Graphics and Image Processing (Theo Pavlidis), Digital Image Warping (George Wolberg), Digital Halftoning (Robert Ulichney), Applied Concepts in Microcomputer Graphics (Bruce Artwick), Photorealism and Ray Tracing in C (Kathleen Clay and Finlay), Practical Image Processing in C (Craig Lindley) y Zen of Graphics Programming (Michael Indexx), tablas de contenido, etc.

Precio de venta: **8.621** pts
10+*: 7.328 pts



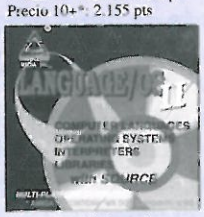
Más de 6.800 ficheros, con código fuente, utilidades y documentación para la serie de compiladores Turbo de Borland, incluyendo C, C++ y Pascal.

Precio de venta: **3.879** pts
Precio 10+*: 2.995 pts

Language/OS

Recopilación de más de 150 lenguajes de programación incluyendo compiladores, intérpretes, librerías, código fuente, lenguajes estándar y experimentales, etc. Para plataformas DOS, Windows, Mac, Unix y Amiga.

Precio de venta: **2.586** pts
Precio 10+*: 2.155 pts



Recopilación de más de 150 lenguajes de programación incluyendo compiladores, intérpretes, librerías, código fuente, lenguajes estándar y experimentales, etc. Para plataformas DOS, Windows, Mac, Unix y Amiga.

Precio de venta: **2.586** pts
Precio 10+*: 2.155 pts

QRZ! Ham Radio

Colección de programas, utilidades, documentación y base de datos para radioaficionados. Incluye la base de datos Callsign.

Precio de venta: **2.241** pts
Precio 10+*: 1.790 pts

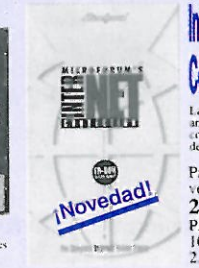
Programmer's Rom

Información y utilidades sobre los más conocidos lenguajes de programación para DOS.

Precio de venta: **2.155** pts
Precio 10+*: 1.690 pts

Las mejores aplicaciones para DOS y Windows clasificadas en 98 temas, desde la agricultura al proceso de textos. Extraída de la librería shareware PDSL. Incluye libro de 96 páginas.

Precio de venta: **4.310** pts
Precio 10+*: 3.715 pts



Internet Connection

Las páginas amarillas completas de Internet.

Precio de venta: **2.586** pts
Precio 10+*: 2.155 pts

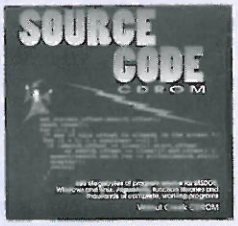
4.4BSD Lite

La última distribución pública BSD de Berkeley. Sólo código fuente.

Precio de venta: **3.448** pts
Precio 10+*: 2990 pts

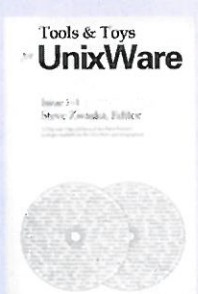
Recopilación de los más innovadores lenguajes de programación y herramientas disponibles hoy en día. Con el sello Dr. Dobb's.

Precio de venta: **6.466** pts
Precio 10+*: 5.603 pts



Sistema operativo de 32 bits FreeBSD 2.0, basando en 4.4 BSD Lite. Incluye código fuente y todo tipo de utilidades y herramientas de conexión y desarrollo.

Precio de venta: **3.017** pts
Precio 10+*: 2.095 pts



Colección de los mejores paquetes y herramientas freeware para UnixWare. Incluye libro de 130 páginas.

¡Novedad!

Precio de venta: **7.759** pts
Precio 10+*: 6.900 pts

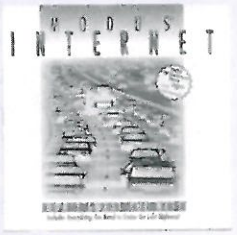
The Internet for Everybody

Cómo usarde personalmente para introducirse y sacar partido de la red. Contenidos de 2 tomos de unos 500 páginas de alto calidad, de todas las secciones en inglés.

Precio de venta: **2.586** pts
Precio 10+*: 2.155 pts

17.240 documentos sobre Internet, incluyendo FAQs, RFCs, mapas, discusiones técnicas de Usenet, tipuntos estándards IEEE, ISO, etc.

Precio de venta: **2.586** pts
Precio 10+*: 2.090 pts



Colección de las mejores utilidades, programas y documentación para acceder a Internet. Entornos DOS, Windows y Macintosh.

Precio de venta: **2.241** pts
Precio 10+*: 1.690 pts

Código fuente principalmente para sistemas UNIX, aunque muchos paquetes pueden compilarse y usarse bajo otros sistemas operativos. Más de 600MB.

Precio de venta: **2.586** pts
Precio 10+*: 2.155 pts

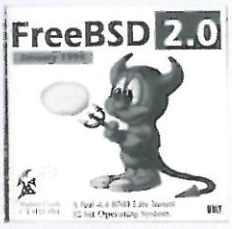
Recopilación completa de los grupos de news de Usenet con sources y alt sources, además de FAQ y código fuente de sid-units.

¡2 CDs!

Precio de venta: **2.586** pts
Precio 10+*: 2.155 pts

Binarios y código fuentes de las implementaciones UNIX para PC's basados en 386/486, de FreeBSD 1.0 y FreeBSD 2.0.

Precio de venta: **2.155** pts
Precio 10+*: 1.690 pts



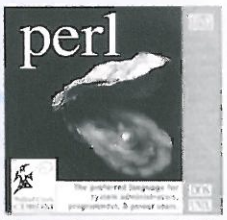
Sistema operativo de 32 bits FreeBSD 2.0, basando en 4.4 BSD Lite. Incluye código fuente y todo tipo de utilidades y herramientas de conexión y desarrollo.

Precio de venta: **3.017** pts
Precio 10+*: 2.095 pts

Internet Surfer

Colección de programas y utilidades para introducirse en Internet. Entornos DOS y Windows.

Precio de venta: **2.241** pts
Precio 10+*: 1.690 pts



El lenguaje y entorno de trabajo preferido por los administradores del sistema, programadores y usuarios avanzados. Incluye código fuente y binarios. Para sistemas Amiga, Atari, AUX, LynxOS, Mac, MPE, MS-DOS, MVS, NetWare, Windows/NT, OS/2, VMS y Xenix.

Precio de venta: **3.879** pts
Precio 10+*: 2.995 pts

Documentación sobre estándares domésticos e internacionales. Recomendaciones CCITT/ITU, RFC's e IEN's, ISO, Winsock, IEEE 7 IESG, etc.

¡2 CDs!

Precio de venta: **2.586** pts
Precio 10+*: 2.155 pts

Productos importados y distribuidos por...

abc analog, s. l. *Directo*

☎ (91) 6342000 / 6343213
(91) 6344786 / 6341293
Apartado de correos 157
28220 Majadahonda (Madrid)
Módem: (91) 6345100 / RDSI (91) 6347020

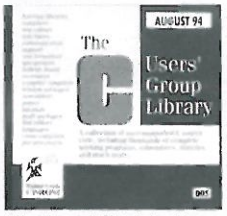
IVA no incluido en los precios
Precios válidos salvo error tipográfico y sujetos a cambio sin previo aviso
Todas las firmas y títulos mencionados están registrados por sus respectivos propietarios
*Consultar condiciones

VISA

C/ Arapiles 7 Esc. D bajo A, P.I.C. 28220 Majadahonda (Madrid)

Las mejores aplicaciones para DOS y Windows, clasificadas en 98 temas, desde la agricultura al proceso de textos. Extraída de la librería shareware PDSL. Incluye libro de 96 páginas.

Precio de venta: **4.310** pts
Precio 10+*: 3.900 pts



La librería de usuarios de C hasta el volumen 411, cientos de megabytes de código fuente y documentación sobre C. También contiene los listados de código fuente originales publicados en The C Users Journal.

Precio de venta: **3.879** pts
Precio 10+*: 2.995 pts

Utilidades, código fuente y drivers para sistemas Unix y PC's bajo DOS y Windows.

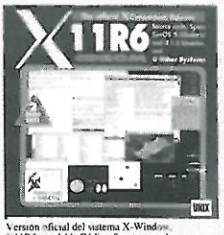
Precio de venta: **2.586** pts
Precio 10+*: 1.983 pts

Incluye el lenguaje de utilidad perl, ideal para administración de sistemas, la herramienta y lenguaje de comandos tcl, y el kit de herramientas para X Window tk.

Precio de venta: **2.585** pts
Precio 10+*: 2.155 pts

Colección de utilidades de programación

Precio de venta: **2.155** pts
Precio 10+*: 1.690 pts



Versión oficial del sistema X-Window, X11R6 nivel 11. Código fuente completo, precompilado para numerosas plataformas UNIX.

¡2 CDs!

Precio de venta: **3.879** pts
Precio 10+*: 2.990 pts

ENTREVISTA

ANTONIO LINARES

En éste artículo Sólo Programadores entrevista a Antonio Linares, creador de las librerías FIVEWIN. Estas librerías, el producto informático español de mayor difusión, posibilitan la codificación de aplicaciones Windows en Clipper. Su autor nos habla de su desarrollo, prestaciones y difusión.

¿ Cómo te iniciaste en el mundo de la programación ?

Al igual que muchos otros programadores, pertenezco a la generación del ZX-81 y el ZX-Spectrum. Aquel milagro de teclas de goma me descubrió la maravilla de la programación.

A continuación pasé al primer IBM-PC. Recién aparecido en España y con un precio totalmente inasequible. Me llevaba los manuales del PC a mi casa "a escondidas", para ver si me aclaraba con aquello del MSDOS. Un día, en un sólo disquete de 5 1/4, alguien me pasó un programa que me condicionaría definitivamente: el dBase II.

Soy 100% autodidacta. Nunca he ido a una facultad de informática, ni a academias ni a nada parecido. Me he dedicado a "devorar" todos los libros y revistas de programación que caían en mis manos, y a programar como un loco tanto como he podido.

Del dBase II, pasé al dBase III, dBase III+, FoxBase+, dBFast y finalmente Clipper. Y con Clipper ya para siempre... Mis lenguajes habituales desde hace años son Clipper y C.

¿ Cuándo se te ocurrió la idea de crear estas librerías? ¿ Cuánto tardaste en finalizar la primera versión ?

Todos los que programamos habitualmente en Clipper hemos echado siempre en falta el poder trabajar en modo gráfico. Por supuesto Windows representa lo máximo. Trabajar en Windows actualmente es obtener muchísimo más rendimiento del ordenador. Una vez que te acostumbras a Windows, el MSDOS queda como "restos" del pasado. Ya no vuelves nunca más a él. El presente y el futuro está en Windows.

Clipper es una herramienta sencillamente extraordinaria. Si hacemos una distinción entre los lenguajes que generan código nativo, y aquellos que trabajan en modo interpretado, está clarísimo que los estándares actuales son el lenguaje C en nativo, y Clipper en pcode. Es por esto que la idea de tener Clipper en Windows ha sido durante mucho tiempo una obsesión, que al final dio vida a FIVEWIN.

Hemos trabajado muchas, muchísimas horas para hacer y desarrollar FIVEWIN. El primer prototipo tardó relativamente poco, menos de un año. Y sin embargo, ya llevamos unos tres años sin poder hacer otra cosa que no sea FIVEWIN.

¿ Trabajaste solo o contaste con la ayuda de otros programadores?

Francisco Pulpón es mi compañero infatigable de aventuras. Es la única persona con la que puedo hablar de lo que quiera sin miedo a que nos tomen por locos. Cuando trabajamos juntos corremos el riesgo de no dormir en toda la noche, dándole vida a todo tipo de ideas y conceptos de programación. Hay tantísimo todavía por hacer...

Para llegar a hacer algo importante hay que entregarse totalmente. En cuerpo y alma. Es la única manera de entrar en sintonía con lo que quieres. Puedes entenderlo como pasión, o locura, es el precio de poder transformar un sueño en realidad.

FIVEWIN es el producto informático español de mayor difusión. ¿Qué sientes al pensar que es creación tuya?

Actualmente FIVEWIN se distribuye principalmente en EE.UU. y Europa, aunque también tiene una presencia notable en Sudamérica, Rusia, Taiwan, Hong Kong, Corea del Sur, Sudáfrica, Australia e Israel. Es el único producto software español que se anuncia a página completa, en la revista más importante de Estados Unidos de desarrollo de bases de datos "Data Base Advisor".

Por supuesto que esto nos hace sentirnos muy orgullosos, aunque también te diré que día a día aumenta la responsabilidad que todo esto conlleva.

Cuando muchas empresas confían en tu producto sabes que tienes que darles lo mejor. Y eso se traduce en más y más horas de trabajo.

El dinero que se obtiene actualmente de las ventas se está reinviertiendo en publicidad y promoción. No hemos contado con ninguna ayuda económica ni subvención de nadie, algo que no nos vendría nada mal.

También es chocante que nadie en España sepa de nosotros. Ni la Junta de Andalucía, ni el Ministerio de Industria, ni las revistas españolas especializadas. España podría cultivar una verdadera cantera de los mejores programadores y el mejor software internacional, algo que generaría riqueza y puestos de trabajo. Y sin embargo, muchas, muchísimas veces nos preguntamos que en qué estarán pensando.

¿Por qué, elegisteis la vía del shareware para su comercialización?

Con la terrible competitividad que existe hoy en día sólo existen dos alternativas: David o Goliat. Microsoft, Borland, CA... son Goliat. Así que tú tienes que ser David y tener buena puntería con la "honda".

El shareware es algo fantástico. Es barato y superefectivo. Pero también hoy en día existen millones de productos que intentan destacar como shareware. Al final, realmente sólo existe una verdadera arma: la calidad. Tienes que ofrecer lo mejor. Tienes que poner tanto valor añadido como puedas, para que tu producto tenga una posibilidad entre mil.

El software es cultura. Algún día el software será gratuito. Representará la expresión más importante de cultura de nuestra sociedad. Y en todo caso, la gente sólo pagará por el uso de determinados programas, pero se adquirirán de forma gratuita. Se pagarán las horas de uso, y si algo no se usa, si no nos vale, no tendremos por qué desembolsar dinero. Algo así como hoy en día pagamos por la electricidad o el agua. Se paga el uso, no la propiedad.

La piratería tanto como la lucha contra la piratería son el fruto de un planteamiento totalmente equivocado. Nadie puede comprar todo el software que hoy en día se genera en el mundo, nadie tendría dinero suficiente, y sin embargo todos necesitamos de esa cultura para que algún día los ordenadores puedan "transformar" esta sociedad. Las empresas de programación y los programadores necesitamos obtener beneficios para poder invertir en más investigación y nuevos productos. Es por esto que el concepto actual del software tiene que cambiar radicalmente.

El shareware es la primera aproximación efectiva a la realidad del software. Del shareware tenemos mucho, muchísimo que aprender. Aunque esto ya lo saben los "grandes". La asociación del shareware con el medio CDROM puede transformarlo todo.

El software es un mundo distinto, no existen camiones, ni fronteras. En sólo unas horas a través del teléfono un producto puede dar la vuelta al mundo, y entrar directamente en el domicilio del cliente. Realmente el software va a ser uno de los principales agentes para ayudarnos a conseguir una conciencia de "ciudadanos del mundo". Entre los programadores cada vez más van cayendo las nacionalidades. Se respeta la cultura, el talante, la inteligencia, el trabajo, ya no importa dónde has nacido o dónde vives. Como dice IBM, "un sólo planeta, y muy pequeño"

¿Qué ventas habéis realizado hasta el momento?

Relativamente poca cosa comparado con el trabajo y el tiempo invertido. Sin embargo, tenemos confianza en que poco a poco iremos creciendo más y más. Es muy difícil saber exactamente cuánta gente está utilizando FIVEWIN. Muchos programadores lo usan sin haberse registrado, aunque con el tiempo se convencen de su utilidad, y deciden registrarse. Actualmente debemos estar en torno a 8.000 ó 10.000 usuarios en todo el mundo. Que se hayan registrado: muchísimos menos, pero tiempo al tiempo...

¿ Fue rápida su distribución ?

Realmente sí. Es curioso cómo los programadores se guían mucho más de su intuición y de su experiencia que de la "grandiosa" publicidad de las revistas. El marketing miente mucho -algo que alguna vez debería regularse- y, afortunadamente la gente ya no se cree lo que le cuentan. Así que deciden probar por sí mismos, y es ahí donde tienes tu verdadera oportunidad.

La publicidad boca a boca es también muy efectiva. Cuando algo funciona, el programador no puede ocultar su entusiasmo, los resultados se ven, y es así como poco a poco los productos se difunden. Como ejemplo piensa en Clipper, en España prácticamente nunca se anunció, y sin embargo un 80% de la empresas españolas de desarrollo de software trabajan con Clipper.

El mercado más difícil ha sido Estados Unidos. A veces sentíamos que existía un cierto "complot" para impedirnos entrar en él. Es un mercado que se protege mucho. Además, el americano tiene un gran complejo de superioridad que hoy en día sólo es un estorbo. Sin embargo, hay que reconocer que siguen siendo los mejores en muchos aspectos. Quizá la diferencia fundamental está en la gran mentalización que ellos reciben. En España realmente necesitamos desarrollar una conciencia de confianza en nosotros mismos. España está subiendo muchísimo de nivel, pero necesitamos más unión, más conciencia de equipo. Al español le va más la pelea y la envidia. Ese es nuestro único lastre.

¿ En qué mercados se ha asentado con más fuerza ?

En Estados Unidos y en Alemania es en donde mejor se nos conoce ahora y en donde más se nos respeta. ¿Los peores mercados? Nuestro "vecinos" Inglaterra y Francia. Parece que les duele que sus "hermanos menores" de España sepan hacer algo. No quiero decir que eso sea una actitud general, pero es algo palpable.

Es gracioso, en Holanda se llegó a publicar un artículo en donde se hablaba de la modernidad de España que había dado lugar a productos como FIVEWIN. Es cómo si no se creyesen que aquí podemos hacer este tipo de cosas. Supongo que por algo será.

Productos como FIVEWIN están haciendo que se nos empiece a respetar mucho a nivel internacional en desarrollo de productos software. Pero tenemos que hacer mucho más. España tiene que promocionar el desarrollo de

empresas de software. Tiene que apostar por el futuro si quiere tener futuro. ¡Ahora! ¡Ya!

¿ Qué herramientas se emplearon en su elaboración ?

Fundamentalmente trabajamos en lenguaje C. Es el mejor sin duda alguna para la elaboración de este tipo de productos. El C++ está ahí abriéndose camino y conquistando terreno, pero a nivel de desarrollo de herramientas, no está muy aceptado aún.

Clipper es formidable, y al unirlo con el lenguaje C se pueden hacer maravillas.

¿A qué tipo de usuario están enfocadas estas librerías?

Al programador, sin duda alguna. Nosotros nos consideramos "desarrolladores de sistemas". Hacemos herramientas que luego usarán otros programadores. Nuestros productos les ofrecen a las empresas de programación I+D a muy bajo coste y en muy poco tiempo.

Es por esto que antes te hablaba de la conciencia de equipo. Muchas empresas van entendiendo que sólo mediante la especialización y la cooperación tenemos una alternativa válida de futuro. Cuando utilizan FIVEWIN están ahorrándose de tres a cinco años de investigación, y eso se traduce directamente en que pueden hacer mejor software con mucha más facilidad.

Además, les entregamos el código fuente con lo que los programadores pueden aprender y modificar las herramientas que les entregamos. Es la conciencia de equipo. Sin esa unión no hay desarrollo.

Los programadores a su vez nos ofrecen una información importantísima. Tan sólo con escuchar sus comentarios y críticas, puedes mejorar muchísimo lo que estás haciendo. Al escuchar y estudiar sus peticiones y consultas, el producto se desarrolla por sí solo.

Se dice que su periodo de aprendizaje es mínimo y que su facilidad de manejo supera con mucho a otras del mercado ¿Qué otras ventajas ofrece sobre sus competidores?

Para nosotros existe una diferencia fundamental. Nosotros hacemos una distinción entre lo que llamamos "productos de laboratorio", y productos del "mundo real".

Muchas de las herramientas software de más prestigio internacional son excelentes productos de laboratorio, pero no solucionan problemas del mundo real. Si los estudias, por supuesto que reconoces su calidad, pero no sirven para que una empresa pueda ganarse la vida desarrollando software.

FIVEWIN es radicalmente distinto. Esta concebido para que el programador "gane dinero". Para que puedas de verdad abordar un proyecto de programación en Windows en tiempo record. Y que tu cliente quede más que satisfecho.

Tenemos clientes entre los más importantes bancos en España, compañías de seguros, empresas de renombre superconocido a nivel tecnológico, etc... y toda esa gente te aseguro que no se dedica a jugar. Tienen que solucionar problemas y entregar soluciones para ganarse el pan.

Ésta es la razón por la que Clipper se ha hecho tan popular. Puedes firmar un contrato con un cliente y sabes que lo tendrás hecho a tiempo y con todas las garantías. Al usar Clipper con FIVEWIN tienes asegurado el éxito en el desarrollo de aplicaciones profesionales en Windows.

Hace ya tiempo apareció un anuncio de Clipper en Estados Unidos que se me quedó grabado. Se veía un camión llegando al amanecer a una ciudad. Y el slogan decía: "mientras otros te prometen la luna, Clipper entrega sus trabajos a tiempo". Esa es la idea.

Dado que se trata de un producto español ¿ Se dispondrá algún día de documentación en nuestro idioma?

Eso nos lo critican muchas veces: "sois españoles, ¿por qué la documentación está en inglés?"

Nos guste o no el lenguaje de la informática es el inglés. Si quieres hacer algo serio en informática más te vale saber algo de inglés. Si queremos conquistar el mercado de ahí afuera tenemos que saber inglés. Esto es una realidad.

Esto no quita que traduzcamos nuestra documentación al español, de acuerdo. Pero tiempo al tiempo, que no somos ni Microsoft ni Borland.

Muchas veces tenemos que entregar una nueva versión en determinadas fechas, y lo que es un auténtico milagro es el tener la documentación a tiempo aunque sea en inglés.

¿Qué otras aficiones tienes, además de, por supuesto, la programación?

El tiempo que nos queda, que no es mucho, lo dedicamos íntegramente a nuestras respectivas mujeres e hijos. Mi hija, a sus 3 años, ya maneja Windows 95 de fábula. Supongo que ellos serán los verdaderos protagonistas. La nueva generación que se está criando jugando con el ordenador. Para ellos el ordenador es como la tostadora del pan o el frigorífico: no le tienen el menor respeto.

La amistad, nada como un buen amigo con el que pasar horas y horas hablando del futuro que viene. Con quien buscar juntos nuevos desafíos y metas. Nuestra familia y nuestros amigos son nuestro mejor tesoro.

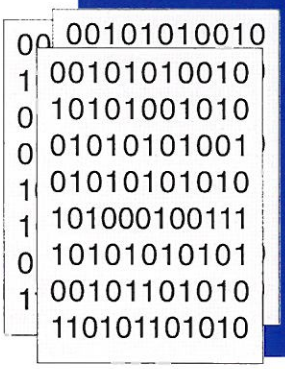
Cuando oigo hablar de los problemas de la droga, del paro, de mucha gente joven que no tiene ideales, siempre pienso que podríamos tener empresas increíbles de desarrollo de software, donde la gente desarrollaría su creatividad y su inteligencia, y en donde se fomentaría el valor de la amistad y el trabajo en equipo. Donde desarrollaríamos soluciones para nuestro pequeño planeta. Donde el dinero no sería un problema sino algo secundario.

Tenemos que transformar el mundo que nos rodea. Ese es nuestro desafío y el ordenador es nuestro aliado. ■

MAIL SIMONS S.L.
Apdo/P.O. Box: 2643
28080 Madrid
Telf: 91-563 44 86
Fax: 91-563 44 51
BBS: 91-563 78 72
CIS: 100342,2763
Internet:bmartine@simons.es

MANEJO SIMPLIFICADO DE SEGMENTOS

Santos Riancho



En los capítulos anteriores se han estudiado distintos componentes y elementos de un programa ensamblador. Se ha visto, entre otras cosas, la forma de codificar procedimientos, implementar sentencias de control y efectuar las operaciones más comunes con interrupciones. Sin embargo, la estructura dentro de la cual se incluían estos elementos, el programa como un todo, no era estudiada en detalle.

Cuando se programa en un lenguaje de alto nivel, la forma real en la que se dispongan las instrucciones y datos de un archivo ejecutable es asunto de los programas compilador y enlazador. En lenguaje ensamblador, cómo no, esto ha de ser dispuesto cuidadosamente por el programador.

En este capítulo se estudiará una manera sencilla de asignar espacio dentro de nuestros programas para el código, la pila y los datos de los mismos, empleando un conjunto de directivas denominadas de control simplificado de segmento. Estas directivas permiten construir cómodamente todo tipo de ejecutables, aunque sin la elasticidad de las "antiguas", llamadas de control completo de segmento. Éstas y las posibilidades que ofrecen se estudiarán más adelante.

ARCHIVOS EXE Y COM

El sistema operativo DOS, sólo reconoce como archivos ejecutables aquellos que poseen por extensiones COM y EXE. Esto significa que, al aparecer uno cualquiera de ellos en la línea de comandos, el DOS "sabe" que su contenido son datos e instrucciones máquina, por lo que le carga en la memoria del sistema, siguiendo una secuencia más o menos compleja, y le cede el control. Los pasos seguidos dependen en gran manera de si el archivo en cuestión es COM o EXE, ya que la estructura interna difiere de un caso a otro. Entender las peculiaridades de esta estructura será fundamental a la hora de manejar las directivas que son tratadas en este capítulo.

EJECUCIÓN DE ARCHIVOS COM

Un archivo de este tipo se caracteriza por tener el código, los datos y la pila en el mismo segmento físico. Es decir, los registros CS, DS y SS apuntan a la misma posición de memoria, no siendo necesario modificarlos en ningún momento para acceder a las distintas partes del programa. Por otro lado, tener todo el programa en un sólo segmento, significa que éste no sobrepasa los 64 Kbytes de longitud.

Cuando el DOS carga un COM en memoria, efectúa básicamente cuatro acciones:

En primer lugar crea el llamado prefijo de segmento de programa (PSP). Ésta es una estructura de 256 bytes que contiene elementos muy variados, como pueden ser instrucciones para la finalización del programa, direcciones de comienzo de algunas interrupciones y bloques de memoria, y caracteres de la línea de comandos del ejecutable. Este PSP también es creado antes de la ejecución de un archivo EXE, y su estructura en detalle se puede conocer a partir de los títulos indicados en la bibliografía al final del capítulo.

Por lo que respecta a la carga en memoria de un COM, lo interesante es saber que el PSP ocupa espacio dentro del ejecutable, ya que el DOS inicializa los cuatro registros de segmento con el valor de la dirección a partir de la que está situado. Ésta puede ser considerada la segunda acción.

Para programar de forma óptima es necesario conocer la mejor manera de disponer los elementos de un programa dentro del mismo. Esto se consigue por medio de las directivas de manejo de segmento, cuya versión simplificada estudiaremos en este capítulo.

Como consecuencia de los dos hechos anteriores la primera posición "útil" es la situada inmediatamente después del PSP. Esta posición tiene asignado el desplazamiento 0100h (256) dentro del segmento del programa, y el sistema operativo asume que la primera instrucción se encuentra situada ahí. Por este motivo el registro IP es inicializado, en un nuevo paso, con el valor 0100h.

Para finalizar, el DOS introduce en el registro SP el valor FFFEh, con lo que todo el espacio no ocupado por el código y los datos del programa es asignado, salvo dos bytes, a la pila.

```
.MODEL
.CONST
.DATA
.DATA?
.FARDATA
.FARDATA?
.CODE
.STARTUP
.EXIT
END
```

Figura 1. Directivas de control simplificado de segmento.

Una vez cargado el programa se procede a su ejecución, siendo destacable en este modelo que las instrucciones y los datos pueden ser referenciados mediante punteros tipo NEAR, al no ser necesarios varios segmentos para alojarlos. Esto se traduce en una mayor velocidad de ejecución, sólo alcanzada, como veremos, por un tipo de archivo EXE.

El que el proceso de carga de estos archivos pueda ser fragmentado en cuatro pasos, es consecuencia de que el sistema operativo espera encontrar una distribución muy rígida dentro de los mismos: primero el código, después los datos y, por último, la pila. Obviamente es responsabilidad del programador que el contenido del programa esté de acuerdo con las expectativas del DOS.

EJECUCIÓN DE ARCHIVOS EXE

Estos archivos se caracterizan, en primera instancia, por no tener límite en cuanto al tamaño del código y datos que pueden manejar, ya que tanto las instrucciones como las variables del programa pueden ocupar más de 64 Kb. Asimismo, el orden real en que se disponen las instrucciones, las variables y la pila no está predeterminado, como en el caso de los COM, sino que es establecido por el programador. Aún se pueden señalar más características, pero bastan las mencionadas para deducir que el sistema operativo no puede cargar un programa EXE de forma correcta si éste, de alguna forma, no le suministra la información necesaria.

Esta información está contenida en una estructura de datos llamada cabecera (header), que es creada por el programa enlazador (link.exe en el MASM y tlink.exe en el TASM) al tiempo que construye el programa, y colocada al inicio del mismo incrementando su tamaño en 512 bytes. Teniendo esto presente se puede describir la carga de un EXE por parte del DOS de la siguiente forma.

En primer lugar, de igual manera que con los COM, se crea un PSP a partir de la primera dirección física libre múltiplo de 16. Acto seguido DS y ES se inicializan con ese valor, de forma que, si se desea, se pueda acceder rápidamente a la información contenida en él.

Modelo	Código	Datos
Tiny	NEAR	NEAR
Small	NEAR	NEAR
Medium	FAR	NEAR
Compact	NEAR	FAR
Large	FAR	FAR
Huge	FAR	FAR

Figura 2. Modelos seleccionables mediante .MODEL.

El siguiente paso que se da es el de asignar a CS e IP los valores necesarios para apuntar a la primera instrucción a ejecutar. El valor de IP, que puede oscilar entre 0000h y FFFFh, se obtiene de la cabecera, y al de CS se llega tras una serie de cálculos basados en la estructura del programa, la longitud de los distintos elementos que lo componen y la dirección inicial del PSP. Hecho todo esto, queda por resolver la dirección de comienzo de la pila y su tamaño, es decir, los valores iniciales de SP y SS.

En principio estos valores también se obtienen a partir de la cabecera, pero, como se verá más adelante, puede ser necesario que las instrucciones del programa los modifiquen en el caso de desearse la misma dirección de comienzo para el segmento de datos y la pila. Esto último, cuando es posible, permite acceder a las posiciones de un segmento mediante los mismos desplazamientos que poseen en el otro, facilitando, por ejemplo, el paso de parámetros en llamadas a subrutinas.

ALGO MÁS: REALOJAMIENTO

En la anterior descripción se han quedado, temporalmente, algunas cosas en el aire. A medida que avance el capítulo se aclararán la mayoría de ellas, pero no se puede dar por finalizado el esquema de carga del EXE sin mencionar la asignación de direcciones físicas reales a los distintos elementos del programa.

Como ya habrá notado el lector, lo que en un archivo COM era suficiente, esto es, referirse a variables e instrucciones mediante su desplazamiento con respecto al único segmento existente, en un archivo EXE puede ser imposible. Desde el momento en que los datos y/o el código pueden abarcar varios segmentos, se hace imprescindible poder cargar DS o CS con las distintas direcciones de comienzo de aquéllos a la hora de acceder a sus contenidos.

Asignar desplazamientos a los distintos elementos de un programa es una tarea mecánica que realiza el programa enlazador, pero, dado que la posición en que se carga un programa depende del estado actual de la memoria del sistema, es imposible determinar de antemano el valor de segmento necesario para localizar un dato o una instrucción. Este problema es solventado por el enlazador colocando un pseudovalor en cada punto en el que aparece una referencia a cualquiera de los segmentos que componen el programa. A la vez, dentro de la cabecera se depositan las direcciones, dentro del programa, de los puntos en los que aparecen esos pseudovalores. Cuando el archivo EXE sea cargado, el sistema operativo se encargará de cambiarlos por las direcciones apropiadas antes de ceder el control al programa. Este proceso, que permite ejecutar un EXE en cualquier punto de la memoria es conocido como realojamiento o reubicación.

COM VERSUS EXE

Por lo visto más arriba está claro en qué casos será forzoso codificar un programa como un EXE. Ahora bien, si

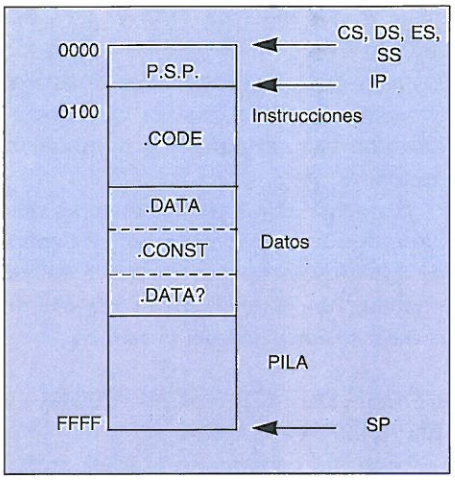


Figura 3. Archivo COM en memoria.

el código, los datos y la pila de nuestro programa no van a ocupar más de 64 Kb ¿será siempre apropiado codificarlo como COM? Veamos los pros y los contras de ambas posibilidades.

Las principales ventajas de un archivo COM frente a un EXE se resumen en dos palabras: compactación y velocidad. Dado que este tipo de archivo no necesita de una cabecera para ser cargado en memoria, ocupa menos espacio en el soporte y es puesto en marcha por el sistema operativo antes que un EXE. Por otro lado, como su estructura es convenientemente sencilla, estos programas son los mejores candidatos para colocar código residente en memoria. Sin embargo presentan una pequeña pega que es obviada de entrada en los archivos EXE.

Se vio en su momento que, cuando se carga un programa COM, el registro SP es inicializado de forma automática con el valor FFFEh. Esto significa que se puede utilizar como pila toda la parte del segmento no ocupada por instrucciones y variables, lo que en muchas ocasiones se traduce en que se dispone de una zona de almacenamiento temporal mayor de la necesaria. ¿Hay algo de malo en ello? La verdad es que no, siempre y cuando detrás de ese programa no vayan a ser cargados otros. Sin embargo, si se está codificando un shell o cargador de programas, es conveniente dejar a éstos la mayor cantidad de memoria posible, por lo que el programador debe efectuar, antes de las habituales operaciones para liberar memoria, un reajuste en el valor de SP, de forma que la pila quede reducida al tamaño imprescindible. Con un archivo EXE este problema no se presenta debido a que es posible especificar durante la codificación cuántos bytes se desean para la pila.

CONSTRUCCIÓN DE UN PROGRAMA

Una vez que estamos en condiciones de determinar el formato apropiado para un programa, es necesario conocer la función de las directivas mostradas en la figura 1. Para ello se desarrollarán paso a paso programas muy sencillos, efectuando en todo momento las aclaraciones pertinentes sobre el significado de las directivas empleadas.

DESARROLLO DE UN ARCHIVO COM

Dado que es el caso más simple, se comenzará describiendo la codificación de un programa COM que visualice el clásico mensaje ¡HOLA MUNDO!. Este programa es el almacenado en el ejemplo TINY.ASM, en el disquete que acompaña a la revista.

```
MOV DX, dirección_segmento_datos
MOV DS, DX

MOV BX, SS
SUB BX, DX

SHL BX, 1
SHL BX, 1          SHL BX, 4 en un 80286
SHL BX, 1            o superior
SHL BX, 1

CLI                No necesario en un 80286
                     o superior

MOV SS, DX
ADD SP, BX

STI                No necesario en un 80286
                     o superior
```

Figura 4. Instrucciones equivalentes a .STARTUP.

En primer lugar se debe informar al programa ensamblador del tipo de archivo que se desea, y eso se hace con ayuda de la directiva .MODEL y mediante la línea .MODEL TINY. Ésta directiva tiene varias finalidades, pero aquí se ha empleado sólo para determinar el modelo de memoria del ejecutable. Con la palabra reservada TINY se indica que el programa va a tener extensión COM. En la figura 2 se muestran los distintos modelos de memoria seleccionables mediante esta directiva.

El siguiente paso es asignar un área para los datos que va a manejar el programa. Esto se logra mediante la directiva .DATA, que indica al programa ensamblador que lo que aparece entre ese punto y la directiva siguiente han de ser, forzosamente, variables. Una instrucción situada en ese área produciría un mensaje de error al ensamblar con el MASM.

Después de establecer un área para los datos, llega el momento de definir el literal que se desea visualizar, con, por ejemplo, la línea MensajeSaludo BYTE '¡Hola Mundo!$'. Que la cadena de texto finalice con el símbolo '$' es debido a que va a ser visualizada con la función 09h del DOS. Esta función necesita de ese símbolo para determinar, dado un punto de comienzo, hasta dónde debe visualizar.

Terminada la definición de los datos del programa es necesario escribir las instrucciones del mismo. Mediante la directiva .CODE indicamos que el área anterior, si la hubiera, finaliza y comienza la correspondiente al código.

PUNTO DE COMIENZO

Sin embargo aún no se puede escribir la primera instrucción del programa. Aunque a nosotros nos parezca obvio, el programa enlazador no va a saber asignar desplazamientos a las instrucciones y variables a no ser que se indique, mediante la directiva .STARTUP, en qué punto exacto comienza el ejecutable. Esta directiva no tiene otra función que la señalada cuando se trabaja con un archivo COM, pero en un archivo EXE será, además, sustituida por una serie de instrucciones muy importantes.

En este momento es cuando comienza la "verdadera" codificación, y con las instrucciones MOV AH, 09h / MOV DX, OFFSET mensajeSaludo / INT 21h se produce la salida del mensaje por pantalla, a partir de la posición actual del cursor.

VOLVER AL DOS

Para finalizar la ejecución y devolver el control al sistema operativo se dispone de la directiva .EXIT, que, programando para DOS es sustituida por las instrucciones MOV AH, 4Ch / INT 21h, que dan por terminado el programa.

Esta directiva no es imprescindible, y el programador puede optar por otras maneras de salir del programa, aunque la asociada a .EXIT es la recomendada por Microsoft para asegurar compatibilidad con futuras versiones. Evidentemente, si se quiere dejar residente parte del código del programa al terminar su ejecución, deberá emplearse la función 31h.

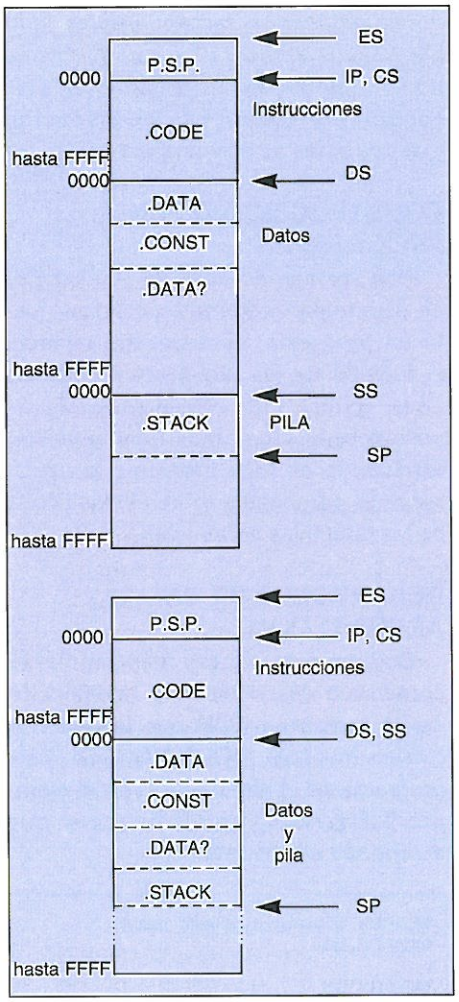
FIN DEL PROGRAMA

Con lo anterior hemos completado casi la codificación del ejemplo. Sólo queda informar al programa ensamblador de que puede dar por concluida su tarea y ceder el control, si es pertinente, al enlazador. La directiva END se encarga de realizar esta función, en el sentido de que todo lo que se escriba a continuación de ella será ignorado en el proceso de traducción a binario del archivo fuente.

DATOS "ESPECIALES"

Además de la directiva .DATA, el lenguaje ensamblador dispone de .CONST y .DATA? para establecer áreas de datos. Con ayuda de la primera es posible definir campos de contenido constante, es decir, variables que van a mantener durante todo el programa su contenido inicial. Esto no significa, ni mucho menos, que el programa no pueda modificarlas en un momento determinado, sino que se señalarán como errores aquellas instrucciones que traten de cambiar sus valores referenciandolas explícitamente. Por ejemplo, si dentro del área marcada por .CONST se define la variable X BYTE 0, la instrucción MOV X, ... dará lugar a un error al ensamblar. Sin embargo, la modificación indirecta MOV SI, OFFSET X / MOV [SI], ... será tratada con toda normalidad, ya que el ensamblador no es tan "listo" como para recordar que SI almacena el desplazamiento de X.

Que quede claro, por lo tanto, que la directiva .CONST no protege nuestros datos de los errores debidos a punteros "desbocados", pero al menos evita los producidos por pequeños despistes. Además aumenta la legibilidad del código al poderse separar, dentro del archivo fuente, datos de distintas características.



Figuras 5 y 5bis. Archivos EXE creados con las opciónes FARSTACK y NEARSTACK.

AHORRO DE ESPACIO

Más tangible es aún el beneficio procedente de definir variables dentro del espacio habilitado con la directiva .DATA?. En cualquier programa se definen variables que no necesitan ningún valor concreto al inicio del programa, como por ejemplo buffers de entrada/salida, campos resultado, etc. Si estas variables son definidas dentro de .DATA contribuyen, sin ninguna necesidad, a aumentar el tamaño imprescindible del ejecutable, que está determinado por los datos inicializados y las instrucciones. En cambio, al declararlas dentro de .DATA?, el tamaño ocupado por ellas en tiempo de ejecución no se refleja en el del programa.

Si se tienen en un programa, por ejemplo, las variables BUFFER BYTE 4000 DUP ? y LINEA BYTE 81 DUP ?, el archivo almacenado en disco verá disminuido su tamaño en 4081 bytes si se emplea la directiva .DATA? para definirlas. Eso sí, el programa enlazador les asignará desplazamientos de forma correcta: aquellos que les correspondan una vez cargado el programa en memoria.

Conocida esta característica, es obvio que será conveniente definir dentro de .DATA sólo las variables que deban contener un valor determinado al comenzar la ejecución del programa.

ORDEN DE LOS SEGMENTOS EN ARCHIVOS COM

Ya se ha comentado que los archivos COM poseen una estructura muy rígida, debido a que deben amoldarse a lo que el sistema operativo espera de ellos. Por este motivo el orden en que aparecen los distintos elementos del programa, instrucciones, datos y pila, debe ser respetado o, llegado el caso, alterado con muchas precauciones.

La única decisión que puede tomar el programador es la de cómo disponer los distintos tipos de datos dentro de la zona de memoria correspondiente a los mismos. El orden habitual, aplicable también a los archivos EXE, es el de .DATA -> .CONST -> .DATA?, independientemente de la secuencia en que hayan sido definidos, pero es posible obtener también el orden .CONST -> .DATA -> .DATA?, intercambiando datos inicializados y de tipo constante. La figura 3 muestra el esquema de un COM cargado en memoria, antes de ejecutarse la primera instrucción.

DESARROLLO DE UN ARCHIVO EXE

Al estudiar la manera de construir un archivo COM, han sido explicadas las funciones de varias directivas. Al pasar al caso EXE, el significado de la mayoría de ellas se mantiene, pero el de algunas es modificado y, además, aparecen otras que no tenían razón de ser en el anterior tipo de ejecutable.

Como se vio en la figura 2, existen distintos modelos de memoria para un programa EXE. Con el fin de no abusar de la paciencia del lector, emplearemos el estudio del modelo SMALL para presentar las características generales de estos archivos, efectuando más tarde los comentarios pertinentes sobre los restantes modelos. De igual forma que

en el caso de los COM, se desarrollará un sencillo programa, SMALL.ASM, para ilustrar los puntos a tener en cuenta cuando se programa usando este modelo de memoria.

CONSTRUCCIÓN DE SMALL.ASM

Como se puede suponer, la primera línea del archivo fuente va a ser .MODEL SMALL. A partir de ella el programa ensamblador "sabrá", llegado el momento, que se ha codificado un programa en el que los segmentos de código, datos y pila pueden llegar a ocupar 64 Kb cada uno.

A continuación se define, dentro de .DATA o .CONST, la variable MensajeSaludo BYTE '¡Hola Mundo!$', de igual manera que en el archivo TINY.ASM, llegando el momento de efectuar una nueva tarea: la de asignar un tamaño a la pila del programa.

La directiva empleada para ello es .STACK seguida de un número que indica la cantidad de bytes que queremos reservar para este área. El número en cuestión puede variar entre 0 y 65536, y su valor por defecto es 1024. En el programa ejemplo el tamaño establecido para la pila es de 64 bytes. Este valor es almacenado por el programa enlazador dentro de la cabecera del archivo EXE, y el sistema operativo inicializa con él el registro SP cuando carga el programa para su ejecución.

Por último, SMALL.ASM acaba con la misma secuencia de directivas e instrucciones que TINY.ASM, desde .CODE hasta END. Sin embargo la similitud es sólo externa porque, como ya se adelantó, en este caso la directiva .STARTUP es sustituida por una secuencia de instrucciones, no limitándose a indicar el punto de entrada del programa.

.STARTUP EN UN ARCHIVO EXE

Cuando un EXE se carga en memoria, el sistema inicializa los registros DS y ES con la dirección del PSP, CS con la dirección del segmento de código, IP con el desplazamiento de la primera instrucción (que SUELE ser 0000), SS con la dirección del segmento de pila y SP con el tamaño de ésta. Excepto los dos primeros valores, el resto se obtiene de la cabecera del EXE.

Por lo que respecta al código y la pila, la situación es correcta. Todo está en orden y el programa puede comenzar. De hecho lo hace, pero existe un "leve" problema: DS no apunta al segmento de datos del programa. Al estar los desplazamientos de todas las variables calculados con respecto al inicio del mismo, habrá que modificar el valor de DS cuanto antes para que éstos puedan ser manejados. Por esta razón las dos primeras instrucciones por las que se sustituye .STARTUP son MOV DX, Dirección_Segmento_Datos y MOV DS, DX. Dirección_Segmento_Datos no es más que una marca que, en el momento de ser cargado el programa, es sustituida por un valor numérico concreto. Este valor se asigna a DS a través de DX, ya que a un registro de segmento no se le puede mover directamente una constante. Es mero convenio la elección del registro DX para esta tarea, AX, habría servido también.

LOCALIZAR LA PILA

Con las acciones anteriores todo está en orden y el programa puede ejecutarse sin problemas. Sin embargo, si el lector edita el ejecutable SMALL.EXE con el comando debug, comprobará que la directiva .STARTUP ha sido sustituida por más instrucciones que las indicadas arriba.

Estas instrucciones, que pueden verse en la figura 4 junto con las que inicializan DS, tienen por objeto reajustar la posición de la pila del programa, de manera que DS y SS apunten al mismo segmento. Veamos a qué responde esto.

La información depositada en la cabecera del EXE acerca de la pila hace que el sistema inicialice SS y SP con valores adecuados, pero asumiendo siempre que datos y pila van a ser segmentos distintos. Esto se traduce en una mayor capacidad de las áreas de almacenamiento del programa, pudiendo cada una de ellas ocupar hasta 64 Kb. Ahora bien, esto no siempre es de interés, y a veces puede resultar más cómodo que ambos segmen-

Update now!

WATCOM

Compiladores para lenguajes profesionales
- Lenguaje C/C++V.10.0 CD ROM
- Lenguaje Fortran 77/32 Bits v 9.5

Ambiente de desarrollo visual (Front End)
- Lenguaje OS/2 Rexx
- VX-REXX v.2.1 Standard Edition
- VX-REXX v.2.1 Client/Server

Bases de datos relacionales (llamada SQL) v 4.0
- Plataformas: Windows, NT, OS/2, DOS y Netware
- Usuarios: Standalone - Network Server
- Lenguaje: C/C++

RAIMA

Raima Database Manager: Base de Datos
- Lenguaje: C
- Sistema Operativo: DOS, Windows, OS/2, UNIX System V, Berkeley 4.2 AIX, Sun OS, SCO, QNX, YULTRIX y VMS
- Plataformas: DOS, OS/2
- Modalidades: Module y System
- Royalty Free

VELOCIS

Velocis: Base de datos. Tecnología Cliente/Servidor
- Lenguaje C/C++
- Sistema Operativo: Windows, NT y UNIX
- Plataformas: Windows, NT, OS/2, Netware 386 NLM, OS/2 y UNIX
- Velocidad de acceso 1/3 más rápido que otras B/Datos

PHARLAP

Gestiona la memoria optimizando los límites de todas las versiones DOS

SEQUITER Software Inc.

Librerías de programas para trasladar aplicaciones de Dbase a C/C++
- CodeServer v.5.1 Cliente/Servidor

OFERTA ESPECIAL

Para los lectores de SÓLO PROGRAMADORES

Infórmate llamando:
Tel.: 485 13 04
Fax: 485 14 29

C. Lope de Vega, 21 bjs.
08005 Barcelona

tos comiencen en la misma posición. Para elegir entre las dos opciones, la directiva .MODEL dispone de las opciones NEARSTACK y FARSTACK. La primera de ellas es la opción por defecto y da lugar a que .STARTUP se expanda de la forma ya vista, de forma que SS y DS adquieran el mismo valor. En el caso de desearse pila y datos aislados se escribiría .MODEL SMALL, FARSTACK al inicio del programa, y .STARTUP se sustituiría, sólo, por las instrucciones que se encargan de inicializar DS. Evidentemente, la opción NEARSTACK no debe emplearse cuando los datos y la pila del programa exceden los 64 Kb.

Hechas estas consideraciones se puede entender el contenido de las figuras 5 y 5bis. Mediante ellas se muestra el mapa de un archivo EXE/SMALL en memoria, después de haberse inicializado DS, SS y SP de forma oportuna, en los dos casos mencionados. Obsérvese que el orden de los distintos segmentos es código, datos y pila, igual que en el caso de los COM. Manejando las directivas de control completo de segmentos es posible obtener otro orden, pero éste es el que siguen normalmente los programas construidos con los compiladores de Borland y Microsoft y, por tanto, el que se seguirá de ahora en adelante.

OTROS MODELOS EXE

Hasta aquí han sido vistas las características más importantes de los programas con modelo de memoria SMALL. Estos pueden considerarse como un paso más en complejidad con respecto a los archivos COM, pero sigue habiendo puntos en común entre ambos tipos: los datos e instrucciones pueden ser accedidos mediante punteros de 16 bits, y la codificación en sí no varía mucho de un caso a otro, siendo, además, tan rápida la ejecución de un archivo EXE como la de un COM.

En cambio, cuando se utiliza el resto de los modelos que vamos a contemplar, es necesario emplear punteros de 32 bits, punteros FAR, para referirse a las instrucciones y/o los datos del programa. Un programa que emplea el modelo MEDIUM puede tener varios segmentos de código, por lo que, quizá, necesita expresar el emplazamiento de alguna instrucción mediante su dirección física completa. De igual forma, si el modelo utilizado es COMPACT, se dispone de varios segmentos de datos. Por último, los modelos LARGE y HUGE, que en la práctica son el mismo, implican el uso de punteros FAR para instrucciones y datos.

MÚLTIPLES SEGMENTOS

De lo anterior se desprende que necesitaremos herramientas apropiadas para definir varios segmentos de datos o código dentro de un programa. Las directivas .CODE, .DATA, .DATA? y .CONST ya contempladas, pueden aparecer varias veces en un programa (aunque NO es habitual), pero el enlazador las agrupa de manera que el conjunto no puede sobrepasar los 64 Kb de código y 64 Kb de datos (modelo SMALL), o 64 Kb de datos y código (modelo TINY). Estas directivas pueden seguir usándose en los restantes modelos, pero resultan insuficientes para nuestro propósito.

La solución, sin embargo, es bien sencilla. Para obtener varios segmentos de código basta añadir a .CODE un nombre de segmento como, por ejemplo, primero o segundo, de manera que .CODE primero y .CODE segundo son considerados ya como dos áreas distintas con un máximo de 64 Kb cada una.

Si lo que se quiere son distintos segmentos de datos, es necesario emplear las directivas .FARDATA y .FARDATA?. Ambas pueden ir acompañadas, al igual que .CODE, de un nombre de segmento que posibilita la existencia de varias áreas de este tipo simultáneamente. La primera de ellas se empleará para definir datos inicializados, y la segunda para aquellos que no necesiten serlo.

El programa LARGE.ASM muestra cómo se utilizan estas directivas. Se ha elegido el modelo de memoria LARGE porque, al permitir varios segmentos de código y datos, sirve para ilustrar todas las características de los modelos MEDIUM, COMPACT y HUGE. Profundicemos un poco en su contenido.

ORDEN DENTRO DE LARGE.EXE

En este programa se han creado tres áreas de código y dos de datos, amén de una pila de 512 bytes. El primer área de código consiste en una subrutina FAR encargada de visualizar cadenas; a continuación, en la segunda área, se tiene la instrucción NOP (código de instrucción 90h) repetida 65500 veces y, por último, hay un conjunto de instrucciones que inicia el programa, visualiza las cadenas y devuelve el control al sistema operativo. Se ha elegido esta disposición para mostrar por un lado la obligatoriedad de emplear una llamada FAR para invocar a la subrutina y, por otro, que el inicio del programa no tiene porqué coincidir con la instrucción que aparece en primer lugar.

BIBLIOGRAFÍA

Microsoft Macro Assembler Programmer's Guide-, V 6.0 y sucesivas, Microsoft

-Microsoft Macro Assembler Reference, Microsoft Corporation

-Michael Tischer: PC interno 2.0. Programación del sistema. DATA BECKER, 1994.

Con los datos se ha hecho algo similar. Se ha creado un mensaje visualizable en cada una de las dos áreas, y se han rellenado éstas para obtener dos segmentos de memoria completos. Así se fuerza al programa a manipular los datos, para visualizarlos, con punteros de 32 bits. La pila se ha creado con la opción FARSTACK, con el fin de evitar su superposición con los datos. Al ocupar cada área 64 Kb es imposible lograr que DS y SS almacenen el mismo valor sin que choquen los contenidos de ambos segmentos.

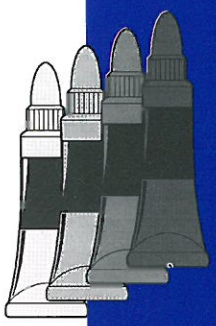
PARA FINALIZAR

Con lo visto hasta este punto se está en condiciones de utilizar las directivas de manejo simplificado de segmento.

Éstas permiten codificar de forma sencilla y completa cualquier programa. No obstante, ya sabemos, que comodidad y total eficacia suelen estar reñidas en el mundo de la programación, por lo que se hace necesario conocer las directivas de control completo de segmento, para estar en condiciones de obtener, en cualquier circunstancia, un código compacto y eficaz. ■

ALGO BIEN DEFINIDO

Agustín Guillén



Con este artículo damos por concluida la serie de trabajos sobre formatos gráficos que llevamos unos meses publicando. Claramente faltaba uno, el más completo en su definición y el más especificado en documentaciones técnicas oficiales. Son tantas sus variantes que muchos desarrolladores no lo eligen por lo complejo de su manejo, pero si sólo se contemplan las más habituales, se simplifica enormemente.

Aunque desde hace algún tiempo ya esta "publicada" la revisión 6 del formato TIFF, su difusión está muy limitada todavía, y este articulo se basará en la popular revisión 5 (aunque ciertos programas actuales aún mantienen sólo compatibilidad con versiones anteriores).

ESTRUCTURA

Como su propio nombre indica, TIFF, su estructura se basa en campos individuales, identificados mediante tags o códigos, uno exclusivo para cada tipo.

Todo fichero TIFF tiene una cabecera de 8 bytes, la cual apunta a uno o varios directorios de imagen. Su forma es la siguiente:

Bytes 0-1.- Especifica el orden de los bytes utilizado en el fichero. Sólo puede ser uno de los siguientes valores:

Hexadecimal	ASCII
4949	II
4D4D	MM

En el orden II (claramente viene de Intel) el orden de los bytes es del menos significante al más significante, ya sea con valores de 16 o de 32 bits. El orden MM (Motorola) es al contrario, del más significante al menos significante, o lo que es lo mismo, tal como escribimos un valor (en hexadecimal, claro).

Bytes 2-3.- Teóricamente indica el número de la versión TIFF, pero por causas fuera del control de los meros programadores, siempre es (y será) 42, o 2A en hexadecimal. Se mantuvo así para evitar que algún programa antiguo diera errores con ficheros TIFF modernos, cuando en realidad era posible su lectura.

Bytes 4-7.- Contiene el desplazamiento en bytes, desde el inicio del fichero, para acceder al primer directorio de imagen. Su único requisito, respecto a su ubicación, es que esté en una dirección par.

Una vez localizado el directorio de imagen, se pasa a interpretarlo. Consiste en un contador de 2 bytes, indicador del número de entradas o campos que contiene el directorio. A continuación están las entradas, cada entrada consta de 12 bytes, y al final de estas aparece un puntero al siguiente directorio de 4 bytes; si no hay más directorios, deben estar a 0. Claramente cada directorio de imagen define a una imagen completa, permitiendo realizar ficheros TIFF multi-imagen.

El formato de cada entrada o campo es el siguiente:

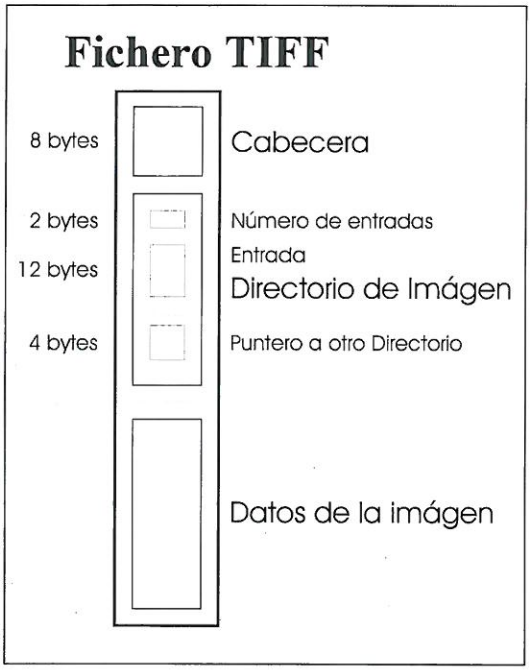
Bytes 0-1.- Tag o código del campo. En el archivo LISTA.DOC se encuentra la lista completa de códigos admitidos en la revisión 5.

Bytes 2-3.- Tipo de campo. Véase en la figura 1 los tipos válidos de datos.

Bytes 4-7.- Longitud del campo. Especificada en unidades del tipo de datos del campo, no en bytes totales. Un sólo LONG tendría una longitud de 1, no de 4.

Bytes 8-11.- Desplazamiento en bytes para acceder a los datos del cam-

Para terminar, se describirá el formato más complejo de todos. Su estructura no es muy enrevesada, pero admite tantas variaciones en su composición, que realizar una aplicación que lea "casi todos" los ficheros de ese formato, es tarea ardua. Se trata de los ficheros TIFF.



Estructura de un fichero TIFF.

po. Debe ser un número par. Si el tamaño de los datos es menor o igual a 4 bytes, entonces no es almacenado el desplazamiento para acceder a los datos, sino los mismos datos. Y si son menores de 4 bytes, son ajustados a la izquierda, dentro de su espacio. Por ejemplo, si se almacena un WORD sólo ocupará los bytes 8 y 9. Para averiguar el tamaño de los datos, se debe consultar el tipo de campo (bytes 2 y 3) y la longitud del campo (del byte 4 al 7).

Las entradas deben estar ordenadas dentro del fichero en función su tag o código, (empezando por el número más bajo).

DESCRIPCIÓN DE LOS PRINCIPALES CAMPOS

Hay un grupo de campos que son imprescindibles para la definición de una imagen en formato TIFF (aunque pueden variar según el tipo de gráfico a almacenar). De cada uno se va a describir su nombre, su código, el tipo al que pertenecen y el número de datos que suelen contener.

BitsPerSample
Código.- 258 (102)
Tipo.- SHORT
Número.- SamplesPerPixel

Define el número de bits por componente de color. En una imagen monocroma, coincidirán los pixels y los componentes, pero si nos referimos a una en color, para obtener un pixel, harán falta varios componentes. Por ejemplo, un gráfico RGB tendrá 3 componentes por pixel y una línea de la imagen se compondrá de 3 líneas conteniendo sus componentes rojo, verde y azul.

Un valor habitual en este campo es 8,8,8, que definiría a una clásica imagen RGB true color.

ColorMap
Código.- 320 (140)
Tipo.- SHORT
Número.- 3 * (2**BitsPerSample)

Describe la paleta de color para imágenes RGB basadas en mapa de color. Cada componente de la paleta se almacena como un SHORT por lo que sus valores oscilan entre 0 (sin intensidad) y 65535 (intensidad máxima). El color negro se representará como 0,0,0 y el blanco como 65535, 65535, 65535.

ColorResponseCurves
Código.- 301 (12D)
Tipo.- SHORT
Número.- 3 * (2**BitsPerSample)

Aquí se definen 3 curvas de color (una para cada componente) cuya longitud, al igual que en el mapa de color es de 2 elevado a los bits por componente de color(2**8 = 256 SHORT para una imagen de 8 bits o 256 colores). Su fin es retocar o corregir la paleta definida en el código 320. Por defecto se toma la curva basada en el sistema NTSC con una corrección del gamma de 2.2.

Compression
Código.- 259 (103)
Tipo.- SHORT
Número.- 1

Indica el tipo de compresión utilizado en el fichero.

1 = Sin compresión. Para calcular el número de bytes por fila se leen ciertos campos y se aplica la formula: (ImageWidth * SamplesPerPixel * BitsPerSample + 7) / 8 si PlanarConfiguration es igual a 1 o (ImageWidth * BitsPerSample + 7) / 8 si PlanarConfiguration es igual a 2. Siempre utilizando aritmética de enteros.

2 = Compresión CCITT Grupo 3 (1-Dimensional Modified Huffman run length encoding).

5 = Compresión LZW. Utilizable para imágenes en escala de grises, con mapa de color y true color.

932773 = Compresión PackBits, un sencillo tipo de compresión RLE para imágenes en blanco y negro.

El valor por defecto es 1 (sin compresión).

GrayResponseCurve
Código.- 291 (123)
Tipo.- SHORT
Número.- 2**BitsPerSample

Su fin es informar con más precisión sobre la interpretación fotométrica de imágenes almacenadas en escala de grises.

Si la GrayResponseCurve de un fichero TIFF es conocida y la de un dispositivo de salida también, entonces será fácil calcular la mejor relación entre ambas de cara a obtener una optima salida de la imagen.

Se utiliza en conjunción con el campo GrayResponseUnit para conocer la precisión de sus valores. Una vez aplicado dicho campo, se suelen obtener valores entre 0.0 y 2.0.

El formato TIFF es el más completo en su definición

GrayResponseUnit
Código.- 290 (122)
Tipo.- SHORT
Número.- 1

Modifica los valores del campo GrayResponseCurve.

1 = Cada número representa décimas de unidad.
2 = Representa centésimas de unidad.
3 = Representa milésimas de unidad.
4 = Representa diez milésimas de unidad.
5 = Representa cien milésimas de unidad.

ImageLength
Código.- 257 (101)
Tipo.- SHORT o LONG
Número.- 1
Altura de la imagen en pixels.

ImageWidth
Código.- 256 (100)
Tipo.- SHORT o LONG
Número.- 1
Anchura de la imagen en pixels.

NewSubfileType
Código.- 254 (FE)
Tipo.- LONG
Número.- 1
Reemplaza al campo SubfileType, debido a las limitaciones que tenía la antigua definición. Se trata de una indicación general del tipo de dato contenido en la subimagen correspondiente al actual directorio de imagen. Se compone de 32 bits que actúan como flags. El bit 0 es 1 si el directorio actual define una versión reducida de otra imagen en el mismo fichero TIFF.

El bit 1 es 1 si la imagen es una simple página de un fichero multi-página.

El bit 2 es 1 si la imagen define una máscara de transparencia de otra imagen contenida en el propio fichero TIFF.

PhotometricInterpretation
Código.- 262 (106)
Tipo.- SHORT
Número.- 1
Define como deben ser interpretados los valores de color de la imagen.

0 = Para imágenes B/N y de escala de grises: 0 es considerado como blanco y 2**BitsPerSample-1 es considerado como negro. El campo GrayResponseCurve "sobrescribe" a éste, y debe ser considerado antes. El valor 0 es el habitual para compresión = 2.

1 = Para imágenes B/N y de escala de grises: 0 es considerado como negro y 2**BitsPerSample - 1 es el valor para blanco.

2 = En imágenes RGB. Para cada uno de los 3 colores primarios (R, G, B) 0 representa la mínima intensidad y 2**BitsPerSample - 1 representa la máxima.

3 = Para imágenes de mapa de color o de paleta.

4 = Máscara de transparencia. Se utiliza para definir una región de otra imagen contenida en el mismo fichero TIFF.
Los campos SamplesPerPixel y BitsPerSample deben estar a 1. Los bits a 1 definen el interior de la región y los bits a 0, la parte exterior.

PlanarConfiguration
Código.- 284 (11C)
Tipo.- SHORT
Número.- 1

1 = Indica que los componentes de color que definen cada pixel son almacenados de manera contigua, como si fuera una imagen de un solo plano. Para datos RGB, quedaría de la siguiente forma: RGBRGBRGB...

2 = Los componentes son almacenados en planos separados. Con datos RGB, primero estarían todos los datos R en un plano, los G en otro y los B en otro.

Predictor
Código.- 317 (13D)
Tipo.- SHORT
Número.- 1
Solo se utiliza con la compresión 5 (LZW). Un valor de 1 indica que no se utilizó un esquema de predicción antes de codificar.

ResolutionUnit
Código.- 296 (128)
Tipo.- SHORT
Número.- 1
Se utiliza con los campos XResolution y YResolution.

1 = No se utiliza ninguna unidad absoluta de medida.

2 = Pulgadas.

3 = Centímetros.

RowsPerStrip
Código.- 278 (116)
Tipo.- SHORT o LONG
Número.- 1
Muestra el número de filas por franja. Cuando se utiliza compresión, los datos están organizados en franjas, de cara a acceder rápidamente a porciones de la imagen.

Para averiguar el número de franjas que contiene una imagen, basta con aplicar la siguiente fórmula: (ImageLength + RowsPerStrip - 1) / RowsPerStrip. Una elección para que cada franja sea de unos 8 Kb es lo más acertado, incluso si la imagen no está comprimida, pues esto limita a un buffer de 8 Kb el espacio necesario para leerlo.

SamplesPerPixel
Código.- 277 (115)
Tipo.- SHORT
Número.- 1
Es el número de componentes por pixel.

1 = Imágenes en B/N, en escala de grises y de paleta de color.

2 = Imágenes RGB.

StripByteCounts
Código.- 279 (117)
Tipo.- SHORT o LONG
Número.- StripsPerImage si PlanarConfiguration es igual a 1.
SamplesPerPixel * StripsPerImage si PlanarConfiguration es igual a 2.
El número de bytes en cada franja. Sirve para precalcular el buffer necesario para leer la imagen.

StripOffsets
Código.- 273 (111)
Tipo.- SHORT o LONG
Número.- StripsPerImage si PlanarConfiguration es igual a 1.
SamplesPerPixel * StripsPerImage si PlanarConfiguration es igual a 2.
Informa del desplazamiento en by-

Tipos de Datos

Tipo	Nombre	Tamaño	Descripción
1	BYTE	1 byte	Entero sin signo de 8 bits
2	ASCII		Cadena de códigos ASCII terminada en 0
3	SHORT	2 bytes	Entero sin signo de 16 bits
4	LONG	4 bytes	Entero sin signo de 32 bits
5	RATIONAL	8 bytes	Dos LONG representando a una fracción

Figura 1. Tipos de datos válidos en un archivo TIFF.

tes para la franja actual. Con esto se permite colocar en cualquier posición a cada franja de datos. Este es el campo que permite encontrar los datos propiamente dichos de la imagen.

XResolution
Código.- 282 (11A)
Tipo.- RATIONAL
Número.- 1

Es el número de pixels por unidad de resolución (campo ResolutionUnit) en la dirección X.

YResolution
Código.- 283 (11B)
Tipo.- RATIONAL
Número.- 1

Es el número de pixels por unidad de resolución en la dirección Y.

Además de estos campos importantes, quedan otras posibles agrupaciones que se podrían clasificar en las siguientes:

CAMPOS DE INFORMACIÓN

Permiten almacenar información diversa sobre la imagen almacenada.

Artist	Autor de la imagen.
DateTime	Fecha de creación del fichero.
HostComputer	Nombre del sistema en donde se creó la imagen.
ImageDescription	Descripción de la imagen
Make	Fabricante del scanner, vídeo o similar.
Model	El modelo del scanner o similar.
Software	Nombre y versión del software que generó la imagen.

CAMPOS RELATIVOS A FAX

Son campos utilizados en el almacenamiento de documentos de fax y no son recomendados para la utilización de imágenes "normales".

Compression Amplía los modos de compresión normales, incluyendo varios tipos exclusivos o diseñados para documentos de fax.

Group3Options	Bits indicadores de parámetros para el tipo de compresión 3.
Group4Options	Bits indicadores de parámetros para el tipo de compresión 4.

Campos para el almacenamiento de documentos

DocumentName	Nombre del documento en donde la imagen está incluida.
PageName	Nombre de la página a la que pertenece la imagen.
PageNumber	Número de página dentro de un fichero TIFF multi-página.
Xposition	Desplazamiento entre el borde izquierdo de la página y el borde izquierdo de la imagen en unidades de resolución (ResolutionUnits).
Yposition	Desplazamiento entre el borde superior de la página y el borde superior de la imagen en unidades de resolución.

CAMPOS A "EXTINGUIR"

Se trata de campos que no se deberían utilizar en la actualidad, aunque se mantienen por compatibilidad.

CellLength	Longitud de la matriz para los procesos de suavizado de la imagen (dithering/halftoning).
CellWidth	Anchura de la matriz para los procesos de suavizado de la imagen.
FillOrder	Orden de los datos dentro de un byte.
FreeByteCounts	Número de bytes de cada bloque.
FreeOffsets	Desplazamiento en bytes, para cada bloque.
MaxSampleValue	El valor máximo para cada componente de color.
MinSampleValue	El valor mínimo para cada componente de color.
SubfileType	Indicación del tipo de dato contenido en la imagen.
Orientation	Indica la orientación de la imagen.
Threshholding	Tipo de Threshholding aplicado a la imagen.

Formato	Ventajas	Inconvenientes
PCX	Simplicidad y rapidez	Escasa compresión
GIF	Compresión	Máximo 256 colores
TGA	Simplicidad y estructuración	Escasa compresión
JPEG	Impresionante compresión	Modifica imagen original y complejidad
BMP	Sencillez y estructuración	Escasa compresión
TIFF	Polivalencia total	Complejidad y variedad.

Figura 2. Ventajas e inconvenientes de cada formato.

EJEMPLO PRÁCTICO

En el disco que acompaña a la revista se adjunta una práctica utilidad que muestra todos los tags incluidos en un fichero TIFF determinado. Este programa es muy útil, además de por los evidentes motivos didácticos, por su utilización para verificar y testear la salida de un conversor, o para averiguar los motivos por los que un fichero TIFF da problemas al ser cargado.

RESUMEN

Seguro que después de leer toda la serie de artículos sobre formatos gráficos, una duda asaltará la cabeza de mucha gente: ¿Cuál es el mejor y cuál debo utilizar en mi próxima aplicación? Si han convivido durante todo este tiempo tantos formatos es porque se complementan. Cada uno de ellos tiene un hueco en el complicado mercado del software, y la realidad es que alguno destaca por su rapidez, otro por su sencillez, otro por su elevado índice de compresión, y el último por su completa polivalencia. En la figura 2 se muestran la mejor y la peor faceta de cada uno de los formatos descritos en esta serie. ■

DESARROLLO DE PROGRAMAS

José C. Remiro

Se han visto hasta ahora conceptos básicos sobre algoritmos y programas. Un algoritmo es una descripción del proceso a seguir para resolver un problema. Los algoritmos no tienen porqué estar expresados en un lenguaje en particular, ni tener una estructura determinada, únicamente deben resolver el problema y estar expresados de tal forma que el sujeto que lo vaya a utilizar sea capaz de llevarlo a la práctica.

Se deduce de la anterior exposición, que cualquier lenguaje para describir un algoritmo debe tener en cuenta a la persona o máquina que lo ejecutará. Un problema entre el diseñador del algoritmo y el elemento que lo ejecutará supondría la imposibilidad de ejecución del algoritmo por parte del ejecutor.

ALGORITMOS, PSEUDOCÓDIGO Y PROGRAMAS

Será el analista y/o el programador, las personas encargadas de diseñar el algoritmo, pero se exige además que el algoritmo esté expresado en un lenguaje con unas reglas sintácticas muy rígidas (lenguaje de programación), de tal forma que el ordenador sea capaz de llevarlo a la práctica.

El pseudocódigo es un lenguaje cercano al lenguaje natural y, por tanto, fácilmente comprensible por personas que no han participado en el diseño del algoritmo. Además, un programador que conozca un único lenguaje de programación, pero que conozca el pseudocódigo utilizado, será capaz de la codificación del algoritmo. Esto permite una mejora en el rendimiento de la programación y la participación en el mantenimiento de un programa, a personas que no participaron en la fase de diseño.

El pseudocódigo permite, además, concentrarse al programador en la lógica del programa, dejando a un lado la rígida sintaxis del lenguaje de programación. Otra ventaja no menos importante del pseudocódigo, es la facilidad de traducción a código fuente (algoritmo escrito en lenguaje de programación) sea cual sea el lenguaje procedural elegido. La principal desventaja del pseudocódigo es que no se puede ejecutar, solamente es un paso intermedio entre la idea que se tiene para resolver un problema y el programa, que supondrá la solución al problema.

PALABRAS RESERVADAS

Todo lenguaje de programación define un conjunto de estructuras mediante palabras reservadas. Una palabra reservada indica una construcción específica del lenguaje, y por tanto las palabras reservadas no pueden utilizarse más que con el fin que indique el lenguaje. En Pascal por ejemplo, BEGIN, PROGRAM, VAR, INTEGER, son palabras reservadas, por tener un significado especial para el lenguaje.

Por otra parte, el programador puede y debe definir sus propias palabras como el nombre del programa o los identificadores (nombre de las variables). La construcción de estos nombres vendrá indicada en el correspondiente manual de referencia del lenguaje; por ejemplo, en Pascal un identificador puede contener letras, números y el carácter de subrayado, debe comenzar obligatoriamente con una letra y si se incluye un carácter subrayado, entonces éste debe ir entre dos caracteres que no sean el carácter subrayado. Se

El éxito del desarrollo de un programa de aplicaciones requiere, además de técnica, un alto grado organizativo. La definición de fases en el desarrollo de un programa, tiene como objetivo que el proceso sea más eficiente y que los programas resultantes sean más fiables, flexibles, y fáciles de mantener.

tiene por tanto libertad a la hora de elegir el nombre de las variables pero deben ajustarse a las reglas del lenguaje.

SELECCIÓN DE NOMBRES

La selección de un nombre de variable, aunque parece trivial, es importante. Un programa puede tener una vida muy larga, a través de la cual sufre diversas modificaciones, además pueden haber otras personas interesadas en comprender como funciona el programa. Con el fin de mantener el programa y su legibilidad se recomienda seleccionar los identificadores de tal forma que indiquen la función que realizan. Por ejemplo, una variable que contenga la suma de un conjunto de números en vez de nombrarla A ó S, se puede seleccionar como nombre SUMA, así su propio nombre da una idea de lo que contiene.

Otros elementos de un programa que se deben seleccionar adecuadamente son los literales. Éstos pueden clasificarse en literales numéricos, que podemos asociar con las cantidades numéricas (3.5, 55, 4.0E13...) y los literales alfanuméricos, que suelen ir encerrados entre apóstrofos (') como en Pascal y C, o entre dobles comillas (Qbasic). Es bastante frecuente cuando se comienza a programar confundir la función de un literal con la función de un identificador. Cuando utilizamos un identificador dentro de una expresión estamos haciendo referencia a su contenido, no al nombre de la variable como tal, mientras que un literal "vale lo que vale". Se ilustra a continuación la diferencia entre la utilización de un literal alfanumérico y un identificador: la función write de Pascal (escribe en la pantalla sus argumentos), aplicada a una variable, visualizará el contenido de la variable, no su nombre, mientras que write aplicado sobre el literal alfanumérico 'Hello World', visualizará Hello World. En definitiva las comillas son usadas por el compilador para diferenciar entre un literal alfanumérico y un identificador.

DESARROLLO DE UN PROGRAMA

Lo mas correcto antes de "ponerse a programar", es estudiar en profundidad las funciones que realizará el programa y estructurar el trabajo a realizar. Se puede dividir el proceso de desarrollo de un programa en varias fases.

LA FASE DE DISEÑO

El objetivo de la fase de diseño es transformar una serie de necesidades del usuario en una descripción técnica de un sistema físico que cumpla las especificaciones del usuario. En esta fase es muy importante seleccionar la estructura del sistema (definición de las diferentes partes y cómo se conectan entre sí). Un sistema bien diseñado tiene una estructura jerárquica, en que cada parte de alto nivel se descompone en subpartes más pequeñas. A medida que se avanza, cada componente del sistema se descompone en detalles cada vez más "finos". Llegará un momento en que las especificaciones puedan ser entregadas a los programadores para su implementación en lenguaje de programación.

Antes de "ponerse a programar" lo conveniente es estudiar en profundidad las funciones que debe realizar el programa

Generalmente, un buen diseño cumple las necesidades básicas de los usuarios, tal vez alguna característica deseable del sistema no se habrá podido adoptar, bien por su complejidad, bien por su coste.

El resultado de la fase de diseño, es un conjunto de documentos que permite describir el sistema con suficiente detalle como para permitir a los programadores elaborar una serie de programas que verifiquen las especificaciones. Estos documentos definen las componentes y la estructura del sistema, las entradas de datos y sus fuentes, las salidas y los algoritmos para transformación de las entradas en salidas. Si el sistema involucra a bases de datos, entonces se incluirá documentación acerca de éstas en la fase de diseño.

LA FASE DE PROGRAMACIÓN

Tras la fase de diseño y a partir del estudio realizado en ella, se pasa a la fase de programación. Consiste en expresar los procesos, especificados en la fase de diseño, en un lenguaje de programación. El programa fuente resultante puede ser entonces traducido automáticamente a programa objeto, supuesto que esté libre de errores.

La selección de un lenguaje de programación, tras la fase de diseño, es importante. Dependiendo del tipo de programas que se vayan a desarrollar, podemos seleccionar un tipo u otro de lenguaje, de todas formas, todos los lenguajes procedurales permiten estructuras de control similares, diferenciándose en la facilidad para manipular los diferentes objetos a programar.

Un criterio para seleccionar un compilador (una vez que se ha seleccionado el lenguaje de programación), es su facilidad de uso y el apoyo que preste a la hora de depurar un programa. Actualmente, los fabricantes proporcionan entornos de programación. Se componen de herramientas para la edición, depuración, compilación y enlace de programas, de tal forma que no se tengan que utilizar herramientas externas al entorno de programación para el desarrollo de los mismos.

Si el diseño realizado es adecuado, la programación es un proceso bastante directo. Una vez creado el fiche-

RESTRICCIONES:

1.- Los dos números deben ser enteros
2.- Ambos números deben ser mayores que cero

Algoritmo de Euclides para el cálculo del m.c.d.

paso 1.- Dividir ambos números
paso 2.- Si el resto es cero, el divisor es el m.c.d
en otro caso tomar el divisor y dividirlo por el resto. Ir al paso 1

Figura 1. Restricciones de los datos de entrada y descripción del algoritmo de Euclides.

ro fuente, mediante un editor de texto, o el propio entorno, pasaremos a compilarlo. El compilador comprobará que la construcción sintáctica del programa es correcta, lo que no asegura la total corrección del programa.

Los compiladores a medida que van progresando en el análisis sintáctico del programa suelen, bien pararse en la línea donde posiblemente se encuentra el fallo o bien, realizar un volcado con el resultado final del proceso de compilación, en ambos casos se obtiene información valiosa a la hora de corregir errores: la línea donde se generó el error, un número identificativo del error (está documentado en el manual del compilador) y un pequeño comentario indicando la causa que produjo el error.

Para depurar el programa, es decir, para obtener un código sintácticamente correcto, hay que tener en cuenta que el compilador puede indicar un mensaje de error en una línea determinada, pero la causa puede que no se encuentre en dicha línea, esto se deberá tener en cuenta a la hora de realizar cualquier modificación sobre el código fuente que se está depurando.

FASE DE VALIDACIÓN

Por último, falta la fase de prueba del programa o de validación. En esta fase, se verificará que el programa cumple los requisitos establecidos en la etapa de diseño. El hecho de que un programa fuente al ser compilado no produzca ningún error de compilación, no implica que el programa sea correcto.

Podemos dividir la prueba de un programa en tres etapas:

1.- Prueba de procedimientos o módulos. Cada módulo individual se prueba usando procedimientos o rutinas artificiales para llamar a los procedimientos a verificar.
2.- Prueba de integración. Grupos de programas que se prueban conjuntamente para determinar si el interfaz es correcto. Este proceso puede ser incremental a medida que se desarrolla el programa hasta que se prueba por completo.
3.- Prueba del programa. Implica la prueba del programa en su totalidad.

```
declaración de variables
 dividendo: entero
 divisor: entero
 resto: entero

inicio
 obtener (dividendo)
 obtener (divisor)
 resto = dividendo mod divisor
 mientras resto <> 0
 dividendo = divisor
 divisor = resto
 resto = dividendo mod divisor
 fin-mientras
fin
```

Figura 2. Pseudocódigo del algoritmo de Euclides.

La fase de validación se realiza al tiempo que avanza la fase de programación. Primero se verifica cada módulo del programa por separado, si es que existen, aplicándose una serie de datos de entrada y comprobando que la salida es correcta. Si se detecta algún fallo, entonces se vuelve programar el módulo que falló (de ahí la importancia de dividir los procesos complejos en subprocesos más sencillos). A medida que los módulos van siendo verificados se agrupan, hasta formar la totalidad del programa. Por último se verifica la corrección del programa en su totalidad.

Los errores más usuales que pueden encontrarse durante la prueba de un programa se pueden dividir en errores lógicos y los debidos a la introducción

```
Program m_c_d_1;
Var
dividendo, divisor, resto: integer;

Begin
Write(' Introduzca un número ');
Readln(dividendo);
Write('Introduzca otro número ');
Readln (divisor);
resto:= dividendo mod divisor;
While resto <> 0
Do
Begin
divisor:= dividendo;
dividendo:= resto;
resto:= dividendo mod divisor
End;
Writeln ('El máximo común divisor es .... ', resto)
End.
```

Figura 3. Primera implementación en Pascal del algoritmo de Euclides.

de datos erróneos. Los primeros se caracterizan por que el programa ejecutado sobre una entrada correcta funciona erróneamente, bien ofreciendo una salida inválida, o bien debido a que el programa no progresa adecuadamente (el programa no detiene su ejecución).

Para determinar donde se encuentran los errores de tipo lógico dentro de un programa se pueden utilizar diversas técnicas. La más rudimentaria consiste en visualizar valores de expresiones, mediante una instrucción del propio lenguaje de programación, donde se cree localizado el fallo, dependiendo del valor de tales expresiones se realizan las modificaciones oportunas. Si el compilador está integrado en un entorno de programación, es posible disponer de una serie de herramientas más avanzadas que pueden ayudar a depurar el programa. Así, podemos utilizar la opción (si es que existe en nuestro entorno) de ejecución paso a paso (trace on), o incluir puntos de ruptura (breakpoints). Un punto de ruptura consiste en detener en un punto del programa su ejecución para comprobar el estado del programa (punto de ejecución donde se encuentra, valor de las variables y expresiones en ese punto).

De todas formas, la depuración de este tipo de errores se puede evitar, en gran medida, durante la fase de diseño. También es buena práctica probar cada uno de los módulos o procedimientos antes de añadirlos al programa global. Una buena comprobación de los módulos o procedimientos añadidos, hace que la búsqueda de errores quede restringida a un espacio reducido.

En cuanto a los errores producidos por la entrada de datos con formato erróneo, se pueden evitar construyendo procedimientos de validación de datos de entrada, de aquí que en la fase de diseño sea necesario hacer un estudio de estos últimos. También es posible que un programa controle ciertos errores, tales como intentar abrir un archivo que no existe, o intentar leer una entrada con un tipo de variable que no se corresponde con el formato adecuado. Por ejemplo, en Turbo Pascal existe una función (Ioresult) que devuelve un código numérico correspondiente a una operación de entrada/salida reali-

```
{Versión mejorada del programa que implementa
el algoritmo de Euclides}
Program m_c_d_1;
{*****************************************************
**
** este programa calcula el máximo común de
dos números aceptados por teclado.
Se ha utilizado la siguiente convención, las pala-
bras que comienzan por mayúscula son palabras
reservadas en Pascal o funciones propias, el res-
to las ha definido el usuario

*****************************************************}

Var

 dividendo, divisor, resto: Integer;
{datos intermedios el divisor contendrá el mcd al
final del                              programa }
 num1, num2: Integer;
{estas variables contienen los datos de entrada}

Begin  {comienza el cuerpo del programa}
 Repeat
 Writeln ('Programa que calcula el máximo común
divisor');
writeln;
Write(' Introduzca el primer número ');
Readln(num1);
Write('Introduzca el segundo número ');
Readln (num2);
 Until (num1 > 0 ) and (num2 > 0)

{*****************************************************

las instrucciones anteriores comprueban que los
números introducidos son mayores que cero, esta
era una de las restricciones del algoritmo y que
debe comprobar el programa.

*****************************************************}

 dividendo:= num1;
 divisor:= num2;
 resto:= dividendo mod divisor;
{********* cálculo del máximo común divisor con
datos validados **********}
.While resto <> 0
 Do
 Begin
 divisor:= dividendo;
 dividendo:= resto;
 resto:= dividendo mod divisor
 End,
 Writeln;
{esta función sin argumentos deja una línea en
blanco}
 Writeln ('El M.C.D de ', num1, ' y ', num2, ' es ',
divisor)
End. {fin del programa}
```

Figura 4. Versión mejorada y documentada del programa anterior.

zada por el programa, consultando el valor devuelto por dicha función, podemos discriminar el tipo de error que se ha producido y realizar un tratamiento adecuado (en Qbasic, se puede utilizar la sentencia on error, resume, error, y las variables error, err y erl).

Si las operaciones a realizar sobre el programa son complicadas, durante las diferentes etapas se desarrollará el manual del usuario, que describirá las operaciones a realizar por aquellos que deseen utilizar el programa.

ESTILO Y DOCUMENTACIÓN

La forma de programar es tanto una cuestión personal (dos programadores seguramente codificarían el mismo programa de manera diferente), como una cuestión que afecta al grupo de personas que se relacionan con el programa.

Existen buenas razones para mantener un buen estilo de programación, pues se pueden presentar situaciones en las que una persona desee consultar el código de un programa, que se tenga que depurar un programa o bien se desee modificar los programas para actualizarlos. Para realizar cualquiera de las anteriores tares es necesario que un programa presente una buena documentación y se facilite su lectura.

Un hábito importante en el momento de escribir el código fuente, es incluir documentación detallada dentro del programa, mediante, por ejemplo, comentarios. En los programas muy sencillos no es muy necesario, pero a medida que los programas crecen en complejidad son imprescindibles.

Un comentario en Pascal es un texto encerrado entre llaves { }, en C, los comentarios van encerrados entre /* y */, mientras que en Qbasic, los comentarios son líneas que comienzan por apóstrofo ' ó por la palabra reservada REM.

Los comentarios de un programa, junto con la selección de nombres adecuados de las variables y la indentación, constituyen la autodocumentación de un programa.

No hay una norma general en cuanto a la autodocumentación asociada a un programa, pero se suele incluir en la cabecera de éste por medio de comentarios: el nombre del programa, función que realiza, nombre de los programadores, fecha de creación, número de actualización o versión, estructura de los datos de entrada y salida, así como el conjunto de valores válidos y una lista de módulos usados por el programa.

Se suele incluir también, comentarios en las condiciones asociadas a un bucle, indicando la condición de parada del bucle, si ésta es compleja, en las llamadas a los módulos o procedimientos se puede señalar la función que realizan y la salida que proporcionan.

En cuanto a la longitud de un procedimiento, no es aconsejable que sobrepase la longitud de una página aproximadamente.

La indentación, también es una práctica muy aconsejable, consiste en sangrar el conjunto de sentencias que pertenecen a una misma instrucción, lo que permite comprender más fácilmente la lógica del programa.

Junto con los comentarios, y si el programa "lo merece", se puede realizar documentación detallada, indicando la estructura global del programa, la estructura de los archivos que utiliza, las entradas y las salidas, los procedimientos de tratamiento de errores.

La preparación de una buena documentación es costosa en tiempo y bastante aburrida para los programadores pero es imprescindible para el mantenimiento de un programa complejo.

UN PEQUEÑO EJEMPLO

Para ilustrar el empleo de las buenas costumbres en la programación vamos a plantearnos la codificación de un programa que, mediante el algoritmo de Euclides, obtenga el máximo común divisor de dos números.

En la figura 1 se muestra un esbozo del algoritmo necesario y las restricciones que se aplicarán sobre los datos de entrada, con el fin de evitar errores posteriores. La figura 2 contiene el seudocódigo de ese algoritmo y, por fin, las figuras 3 y 4 muestran dos versiones del programa ya escrito en lenguaje Pascal. Aunque ambas resuelvan el problema, la segunda es más eficaz que la primera, ya que efectúa la validación de los datos de entrada y está convenientemente comentado. ■

TRATAMIENTO DIGITAL DE IMÁGENES (III)

Enrique Coiras

En esta tercera entrega se introducirán los métodos para el manejo de imágenes en color y la técnica del entramado

IMÁGENES EN COLOR

Aunque el proceso mediante el cual nuestro cerebro percibe los colores es un fenómeno psicofisiológico complejo que aún no se comprende en su totalidad, la naturaleza física del color puede expresarse con una base formal fundada en diversas teorías y resultados experimentales.

A consecuencia de la estructura del ojo humano, todos los colores se perciben como combinación de tres colores fundamentales o primarios: el rojo, el verde y el azul (R, G y B, según sus iniciales en inglés). Esto es debido a la presencia de tres tipos de conos en la retina, con sensibilidades diferentes a zonas distintas del espectro. La respuesta del ojo es más acusada en la parte verde del espectro, y los conos azules son muchísimo menos sensibles que los verdes y rojos.

Buscando un estándar, la CIE (Comisión Internacional de la Iluminación) designó en 1931 las siguientes longitudes de onda específicas para los tres colores primarios:

rojo = 700.0 nm
verde = 546.1 nm
azul = 435.8 nm

Estos colores primarios pueden ser sumados para obtener los colores secundarios : magenta (rojo y azul), cyan (verde y azul) y amarillo (rojo y verde). Mezclando los tres colores primarios o uno primario y su secundario opuesto obtenemos el color blanco. Esto se da en los sistemas de color aditivo, como las lámparas, televisores y monitores.

Pero también existen los sistemas de color substractivo, como las imprentas, impresoras o la mayoría de los sistemas de fotografía en color, en los cuales la luz blanca pasa a través de filtros o se refleja en pigmentos para dar el color resultante. En ellos los colores primarios se definen como aquellos que absorben uno de los colores primarios de la luz, y reflejan los otros dos. Así, los tres colores primarios de los pigmentos son el cyan, el magenta y el amarillo (C, M e Y, según sus iniciales en inglés), mientras que los secundarios son el rojo, el verde y el azul. Es fácil ver que la combinación de los tres colores primarios o de uno primario y su secundario opuesto, en un sistema de color substractivo, da como resultado el color negro. Las diferencias entre los dos sistemas de color están recogidas en la figura 1.

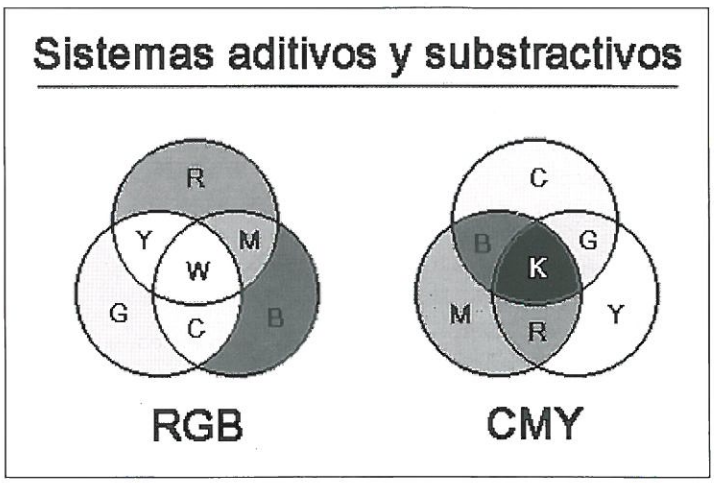


Figura 1 : Combinaciones de los colores primarios en sistemas aditivos y substractivos.

Los monitores y televisores en color son ejemplos representativos de la naturaleza aditiva de los colores luminosos. El interior de los tubos de los televisores en color esta compuesto de

La potencia de los ordenadores domésticos permite realizar funciones de tratamiento de imágenes que hasta hace poco estaban reservadas exclusivamente a las estaciones de trabajo y ordenadores dedicados

grandes matrices de pequeñas triadas de puntos de fósforo, sensibles a los choques de electrones. Cuando es excitado, cada punto de la triada es capaz de emitir luz con una longitud de onda correspondiente a uno de los tres colores primarios. La intensidad de los puntos rojos de la matriz se controla mediante un cañón emisor de electrones que hay dentro del tubo. Lo mismo ocurre con los puntos de fósforo verdes y azules. Una vez representada la imagen en la pantalla el ojo la recibe e "integra" las aportaciones de los tres tipos de puntos, percibiendo al final la ilusión de una imagen de colores continuos.

RGB Y COLOR INDEXADO

Desde el punto de vista práctico, lo interesante de un método de representación del color es que sea fácil de usar en las aplicaciones a las que esté destinado. El sistema RGB es el que utilizan directamente las tarjetas de vídeo para enviarle las imágenes a los monitores. Los modos VGA estándar permiten utilizar paletas de color de hasta 256 colores, cada uno de los cuales está compuesto por tres componentes (rojo, verde y azul) que pueden tomar 64 posibles valores. Así, los valores RGB para los colores primarios serán (63, 0, 0) para el color rojo, (0, 63, 0) para el verde y (0, 0, 63) para el azul. Las componentes de los colores secundarios serán, obviamente, (63, 63, 0) el amarillo, (63, 0, 63) el magenta y (0, 63, 63) el cyan. El (0, 0, 0) representa el negro y el (63, 63, 63) el color blanco. Todos los colores que tengan valores iguales para sus tres componentes serán grises.

Para reservar los colores de la paleta podemos hacer dos cosas. O bien llamamos a las rutinas de la BIOS (interrupción 10H), o bien escribimos directamente en los puertos de la VGA. Llamar a la BIOS no es muy recomendable porque su funcionamiento es bastante lento, así que lo mejor es el segundo método. Para especificar las componentes RGB de un registro de la paleta debemos seguir los siguientes pasos :

1) escribir en el puerto 3C8h el índice del color que queremos modificar
2) escribir el valor de su componente roja en el puerto 3C9h
3) escribir el valor de su componente verde en el puerto 3C9h
4) escribir el valor de su componente azul en el puerto 3C9h

Para rellenar toda la paleta no tenemos mas que repetirlo para cada uno de sus 256 registros. Eso es precisamente lo que hace la siguiente rutina, que lee una paleta de un fichero y la carga :

```
#define PALETTE_SIZE 256
void readpalette (char *filename)
{
  FILE *stream;
  unsigned char palette [PALETTE_SIZE][3];
  int k;

  /* leemos fichero */

  stream = fopen (filename, "rb");
  if (stream == NULL) {
    printf ("\nREADPALETTE : No puedo leer el fichero '%s'.\n", filename);
    exit (1);
  }
  else
     fread ((unsigned char *) palette, 3, PALETTE_SIZE, stream);

  fclose (stream);

  /* volcamos paleta */
for (k = 0; k < PALETTE_SIZE; k ++) {
    outp (0x3C8, k);
    outp (0x3C9, palette [k][0]);
    outp (0x3C9, palette [k][1]);
    outp (0x3C9, palette [k][2]);
  }
}
```

Para leer un registro de la paleta se utiliza un procedimiento similar, solo que en este caso debemos escribir primero en el puerto 3C7h el valor del índice del registro que nos interesa y luego leer, en el 3C9h, cada uno de los valores RGB. La siguiente rutina lee la paleta y la escribe en un fichero :

```
void writepalette (char *filename)
{
  FILE *stream;
  unsigned char palette [PALETTE_SIZE][3];
  int k;

for (k = 0; k < PALETTE_SIZE; k ++) {
    outp (0x3C7, k);
    palette [k][0] = inp (0x3C9);
    palette [k][1] = inp (0x3C9);
    palette [k][2] = inp (0x3C9);
  }
  stream = fopen (filename, "wb");
  if (stream == NULL) {
  printf ("\nWRITEPALETTE : No puedo escribir el fichero '%s'.\n", filename);
    exit (1);
  }
  else
     fwrite ((unsigned char *) pal, 3, PALETTE_SIZE, stream);

  fclose (stream);
}
```

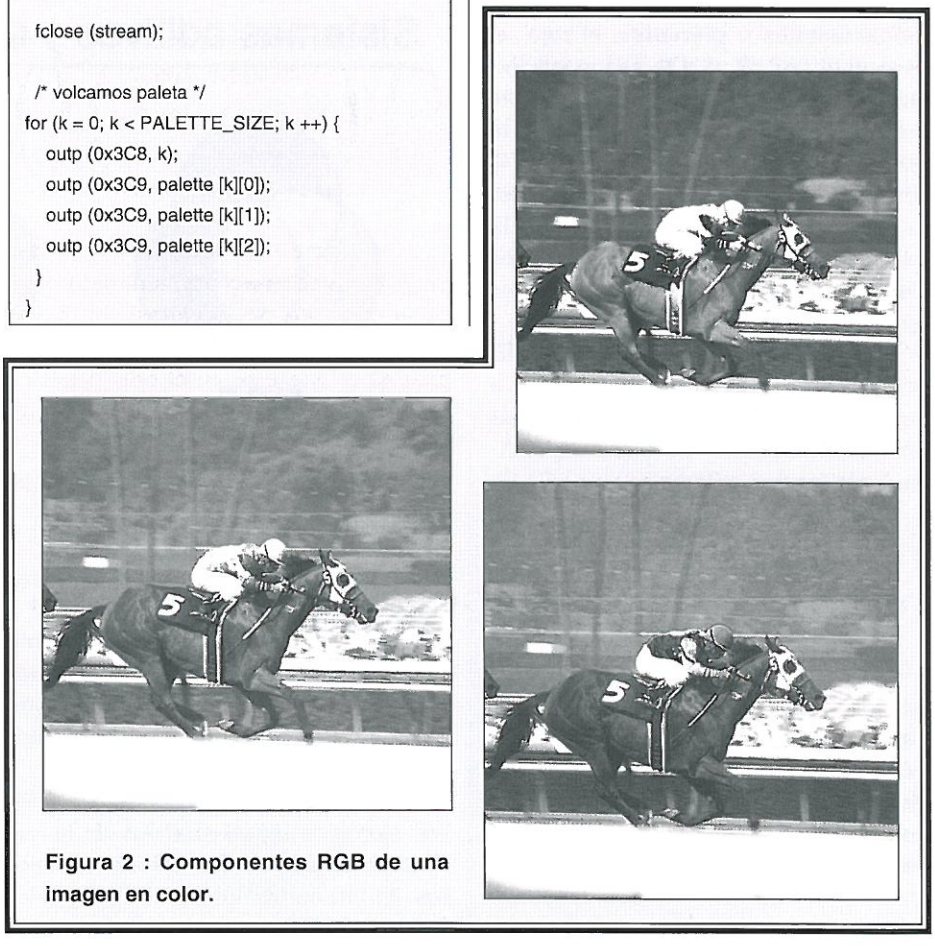


Figura 2 : Componentes RGB de una imagen en color.

De todas maneras, estos modos estándares que utilizan paleta no son verdaderos modos RGB, sino modos indexados (también llamados de pseudocolor), que asocian a cada valor de pixel un color determinado. En un modo RGB real cada pixel está especificado por los valores de sus componentes roja, verde y azul, no por un número que le asocia un color de una tabla.

En las tarjetas gráficas SuperVGA, que pueden manejar hasta 16 millones de colores (modo TrueColor), sí podemos hablar de modos RGB. En el modo TrueColor, por ejemplo, cada pixel viene dado por 3 bytes (24 bits), que especifican las intensidades de cada uno de los colores primarios.

Los modos de pseudocolor se utilizan con frecuencia para resaltar determinadas características de una imagen monocromática. Las típicas imágenes térmicas, en las que las partes más calientes de la escena aparecen en colores rojos y las más frías en azules (como en la película "Depredador"), son un claro ejemplo de utilización de color indexado, ya que las imágenes recogidas por las cámaras térmicas sólo tienen una banda (correspondiente a la zona infrarroja del espectro). También se utilizan estas técnicas en el coloreado de las imágenes de los aparatos de rayos X de los aeropuertos y edificios oficiales para resaltar con determinados colores aquellas zonas de la imagen que pudieran corresponder a armas o explosivos.

SISTEMAS DE REPRESENTACIÓN DEL COLOR

El sistema RGB es de uso obligado cuando queremos comunicarnos con la tarjeta gráfica, pero desde el punto de vista de comodidad de utilización no puede decirse que sea el mejor. El sistema IHS (Intensity, Hue, Saturation) presenta los colores desde un punto de vista más intuitivo. Cada color esta especificado por su intensidad I, que indica como es de brillante; su tonalidad H, que se refiere a la gama de colores a la que pertenece (rojos, violetas, azules, etc); y su saturación S, que indica su pureza, o cantidad de luz blanca que tiene el color. El color rosa, por ejemplo, es un rojo poco saturado. Los colores con saturación cero son todos grises, y los que tienen saturación máxima son todos colores puros.

La función de cambio de base del sistema RGB al HLS viene dada por las siguientes fórmulas :

$$\begin{pmatrix} I \\ V1 \\ V2 \end{pmatrix} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ -1/\sqrt{6} & -1/\sqrt{6} & 2/\sqrt{6} \\ 1/\sqrt{6} & -2/\sqrt{6} & 0 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$H = \arctan (V2 / V1)$$
$$S = (V1^2 + V2^2)^{1/2}$$

Según esta definición, el color azul es el cero para la tonalidad H. La relación inversa viene dada por las transformaciones siguientes :

$$V1 = S * \cos (H)$$
$$V2 = S * \sin (H)$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 4/3 & -2*\sqrt{6}/9 & \sqrt{6}/3 \\ 2/3 & \sqrt{6}/9 & -\sqrt{6}/3 \\ 1 & \sqrt{6} & 0 \end{pmatrix} * \begin{pmatrix} I \\ V1 \\ V2 \end{pmatrix}$$

El sistema CMY (Cyan, Magenta, Yellow o cyan, magenta, amarillo) ocupa el mismo espacio cartesiano que el RGB pero el color blanco (y no el negro) es el que está en el origen (sistema de color substractivo). Es importante conocer el modelo CMY cuando se trabaja con fotocopiadoras, impresoras o impresoras en color, aunque, en muchos casos, además de los pigmentos cyan, magenta y amarillo se utilice también el negro (sistema CMYK). El negro hace que los colores oscuros luzcan más, y oscurece con más eficacia que las mezclas, gastando menos tinta, lo cual es importante porque, además de ser más económico, evita que el exceso de tinta cree problemas de adhesión del papel a la maquinaria de impresión. Las fórmulas para pasar de RGB a CMY y viceversa son:

$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} C \\ M \\ Y \end{pmatrix}$$

En el sistema XYZ se definen unos colores primarios artificiales para conseguir un sistema estándar en el que tengan cabida todos los colores del espectro (existen colores que no pueden conseguirse con una mezcla de las primarias RGB). Los colores primarios XYZ se han elegido de forma que la coordenada Y se corresponda con la luminancia. Las transformaciones entre RGB y XYZ son las siguientes :

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.607 & 0.174 & 0.200 \\ 0.299 & 0.587 & 0.114 \\ 0.000 & 0.066 & 1.116 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.910 & -0.532 & -0.288 \\ -0.985 & 2.000 & -0.028 \\ 0.058 & -0.118 & 0.898 \end{pmatrix} * \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

El sistema Lab se creó para que las medidas de color fueran más sencillas, y también para que fueran independientes del dispositivo en el que se representasen las imágenes, lo cual lo convierte en un sistema ideal para portar imágenes entre sistemas diferentes.

Lab también es el sistema recomendado para imprimir imágenes en impresoras PostScript de nivel 2.

La coordenada L se corresponde con la luminancia (el brillo) y las coordenadas a y b controlan la crominancia (es decir, la información cromática del color en cuestión), lo cual es interesante si queremos trabajar sobre la luminosidad de una imagen sin afectar su color o viceversa. La a controla el rango verde-magenta y la b el azul-amarillo.

Con una trama de patrones binarios nxn podemos conseguir hasta n^2 + 1 niveles distintos

Las transformaciones para los sistemas XYZ y Lab son las siguientes :

```
L = 25 * (100 * Y/Y0)^(1/3) - 16
a = 500 * ((X/X0)^(1/3) - (Y/Y0)^(1/3))
b = 200 * ((X/X0)^(1/3) - (Z/Z0)^(1/3))

X = X0 * (a/500 + (1/100)^(1/3) * (L + 16)/25)^3
Y = Y0 * ((L + 16)/25)^3
Z = Z0 * ((1/100)^(1/3) * (L + 16)/25 - b/200)^3
```

Donde X0, Y0 y Z0 son las coordenadas XYZ del blanco de referencia del sistema en el que nos encontremos.

En la figura 2, podemos ver las componentes RGB de una imagen en color.

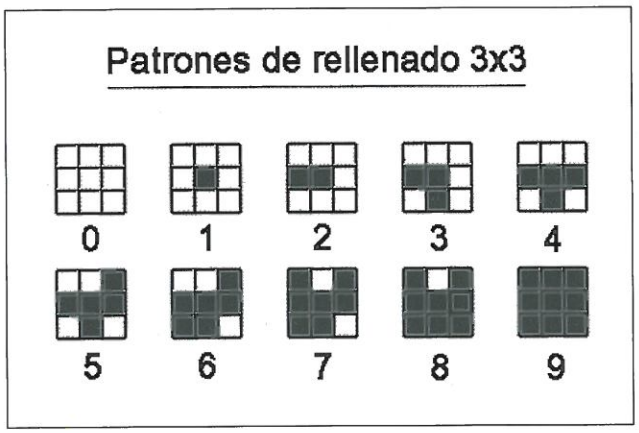


Figura 3 : Patrones de entramado de tamaño 3x3.

UTILIZACIÓN DE TRAMAS

Muchos displays, y sistemas de impresión sólo son capaces de producir dos niveles de intensidad, aunque utilizando tramas podemos extender ese rango. El uso de tramas (*dithering*, en inglés) basa su funcionamiento en la integración espacial que realizan nuestros ojos, que hace que al observar áreas con detalles muy finos desde una distancia suficiente sólo percibamos la intensidad media de la zona.

Este fenómeno se utiliza para la impresión de fotos en los periódicos y libros, usando la técnica de los semitonos. En cada posición de un pixel se imprime un círculo negro cuya área es proporcional a la oscuridad (o sea, 1 menos la intensidad) del pixel de la imagen o foto original. En los periódicos se suele utilizar una resolución de unos 60 u 80 puntos por pulgada, mientras que en los libros y revistas se usan unos 150 puntos por pulgada.

Con nuestra tarjeta de vídeo o nuestra impresora en blanco y negro podemos aproximar los puntos de área variable usados por las imprentas mediante tramas de pequeños patrones. Por ejemplo, un área de pixels binarios (es decir, que solo puedan ser blancos o negros) 3x3 se puede utilizar para representar 10 niveles de gris distintos, como se ve en la figura 3. En general un grupo de nxn pixels binarios puede generar n^2 + 1 niveles de intensidad distintos.

Lo que se está haciendo, en el fondo, es sacrificar resolución espacial para conseguir una mayor resolución en intensidad. El conjunto de patrones utilizado como trama puede representarse por una matriz de enteros que indiquen el "orden de rellenado" de la misma según aumenta la intensidad. Para los patrones 3x3 de la figura 3 la matriz sería :

$$D3 = \begin{pmatrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{pmatrix}$$

Otras matrices de rellenado, para patrones 4x4 y 8x8 son las siguientes :

$$D4 = \begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}$$

$$D8 = \begin{pmatrix} 0 & 32 & 8 & 40 & 2 & 34 & 10 & 42 \\ 48 & 16 & 56 & 24 & 50 & 18 & 58 & 26 \\ 12 & 44 & 4 & 36 & 14 & 46 & 6 & 38 \\ 60 & 28 & 52 & 20 & 62 & 30 & 54 & 22 \\ 3 & 35 & 11 & 43 & 1 & 33 & 9 & 41 \\ 51 & 19 & 59 & 27 & 49 & 17 & 57 & 25 \\ 15 & 47 & 7 & 39 & 13 & 45 & 5 & 37 \\ 63 & 31 & 55 & 23 & 61 & 29 & 53 & 21 \end{pmatrix}$$

Con ellas podemos conseguir 17 y 65 niveles de intensidad, a costa de ver reducida la resolución a 1/4 y 1/8 respectivamente.

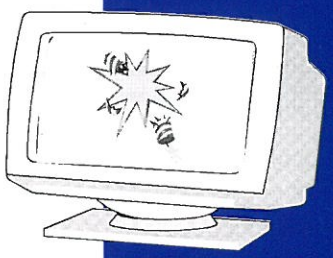
Para ver si un pixel determinado (x,y), debe encenderse utilizando una trama nxn, hay que comprobar que su valor de intensidad en la imagen original sea superior al valor Dn[y mod n][x mod n] de la matriz de rellenado. Si lo es, el pixel se enciende, y si no se deja apagado. A esta técnica de rellenado con tramas se le llama entramado ordenado. La siguiente rutina implementa este método, y convierte una imagen de grises en otra binaria con entramados utilizando la matriz D8:

```
void bindither8 (imagen img)
{
  int y, x;
  int order = 8;
  unsigned char dm64 [8][8] = {
    { 0, 32,  8, 40,  2, 34, 10, 42},
    {48, 16, 56, 24, 50, 18, 58, 26},
    {12, 44,  4, 36, 14, 46,  6, 38},
    {60, 28, 52, 20, 62, 30, 54, 22},
    { 3, 35, 11, 43,  1, 33,  9, 41},
    {51, 19, 59, 27, 49, 17, 57, 25},
    {15, 47,  7, 39, 13, 45,  5, 37},
    {63, 31, 55, 23, 61, 29, 53, 21}
  };

  normalizar (img, 0, (unsigned char) (order * order));

  for (y = 0; y < img.sizey; y ++)
    for (x = 0; x < img.sizex; x ++)
      if (img.dat [y][x] > dm64 [y % order][x % order])
        img.dat [y][x] = (unsigned char) (defnumgrises - 1);
      else
        img.dat [y][x] = 0;
}
```

Esta técnica también puede aplicarse a imágenes en color, solo que para ellas tendremos que hacer tres entramados, uno para cada componente de la imagen. ■

PROGRAMACIÓN DE DEMOS (I): EL FUEGO

Pedro Antón



Como muchos de vosotros sabréis, para hacer una demo son necesarios varios elementos:

- Una o varias canciones. Se usará formato MOD de cuatro u ocho canales.
- Gráficos. Cualquier tipo de dibujo en 256 colores puede ser usado en la construcción de una demo. Será interesante que los motivos de los dibujos sean originales.
- Código. Cualquier efecto que consigáis programar a lo largo del curso podría ser usado en la demo.
- Ideas originales. Casi lo más importante de una demo es la inclusión de efectos nunca vistos hasta la fecha, eso diferencia una demo buena de una del montón.

Dicho esto quiero animar a todos los programadores, músicos, artistas, y lectores en general de Sólo Programadores a que nos envíen sus trabajos a la redacción indicando Ref:CÓMO SE HACE UNA DEMO. Los mejores trabajos serán incluidos en la demo y, por qué no, si estás interesado, tú podrías ser parte de Spanish Lords. Esperamos vuestros trabajos. Además se tendrán en cuenta las peticiones de los lectores sobre determinados efectos, aclarando, en lo posible, la realización y el diseño de éstos.

En este artículo se intentará explicar cómo se hace el bonito efecto del fuego, y espero que disfrutéis viendo como arde vuestro monitor. Para ello se debe tener en cuenta que un factor primordial en cualquier efecto de demo es la velocidad. La mayoría de estos efectos necesitan sacar el máximo partido de la máquina, por lo que antes de empezar se aclararán una serie de cosas.

LA MEMORIA DE VÍDEO

En primer lugar es necesario saber que los accesos a la memoria de vídeo son terriblemente lentos, mucho más lentos que los accesos a la memoria convencional de un ordenador, y que además este tiempo de acceso varía enormemente dependiendo de la calidad de las tarjetas de vídeo. Por ello el afortunado poseedor de una ET4000 en un 386 a 25 Mhz, puede ver algunos efectos más rápido que el poseedor de un 486 a 33 Mhz con una Trident. Intentaremos no usar "modos X", debido a su complejidad y a que estos modos acentúan todavía más estas diferencias.

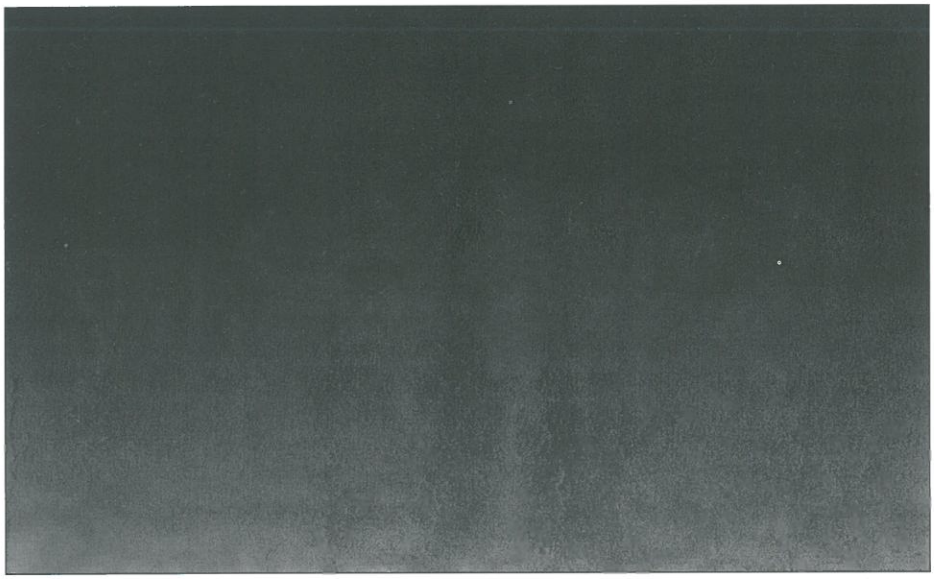
EL BUFFER

En segundo lugar, se debe explicar la importancia del buffer a la hora de programar un efecto para una demo. En algunos casos es evidente que se necesita un buffer si no se quieren ver en la pantalla parpadeos o efectos molestos, pero en otros casos, no es tan claramente necesario. No obstante se empleará para acceder el menor número de veces posible a la memoria de vídeo. Además, transferir un buffer de 64K no es tan lento como parece:

```
MOV AX,0A000h
MOV ES,AX
MOV AX,Segmento_destino
MOV DS,AX
```

El factor primordial en cualquier efecto de demo es la velocidad

Comienza en este número una serie de artículos con el objetivo de mostrar el desarrollo para la realización de una demo. El objetivo final del curso será la construcción de una, y para ello necesitaremos vuestro apoyo y colaboración.



```
XOR DI,DI
XOR SI,SI
MOV CX,32000
REP MOVSW       ;En modo real

MOV AX,0A000h
MOV ES,AX
MOV AX,Segmento_destino
MOV DS,AX
XOR DI,DI
XOR SI,SI
MOV CX,16000
REP MOVSD; En modo protegido
```

De estas líneas de código se debe decir que las instrucciones REP MOVSW y REP MOVSD usan transferencia por DMA y son realmente rápidas, aunque "no es oro todo lo que reluce", ya que aún necesitando la inmensa mayoría de las demos un 386 o superior, la instrucción REP MOVSD no tiene ningún efecto si el bus de datos de la tarjeta de vídeo no es de 32 bits.

Para calcular cada pixel del buffer se calcula la media aritmética de los valores de los pixels que lo rodean

LA PALETA

En tercer y último lugar, hay que acentuar la importancia de la paleta. Sí: algo tan aparentemente simple como una paleta de 256 colores tiene una enorme importancia en gran cantidad de efectos. Las paletas pueden ser construidas de varias formas: programándolas o usando programas de diseño gráfico para posteriormente capturarlas. Es muy importante dar a la paleta la importancia que tiene.

El código para leer un color de la paleta es:

```
MOV   DX,03C7h
MOV   AL,Color
CLI
OUT   DX,AL
MOV   DX,03C9h
IN    AL,DX
MOV   R,AL
IN    AL,DX
MOV   G,AL
IN    AL,DX
MOV   B,AL
STI
```

y para escribir un color en la paleta:

```
MOV   DX,03C8h
MOV   AL,Color
CLI
OUT   DX,AL
INC   DX
MOV   AL,R
OUT   DX,AL
MOV   AL,G
OUT   DX,AL
MOV   AL,B
OUT   DX,AL
STI
```

ELEMENTOS NECESARIOS

Explicado esto, se procederá a ver qué se necesita para hacer un fuego; ¿un buffer?, claro que sí, pero ¿de 320x200?, no, eso sería demasiado lento y ni siquiera más bonito. Se trabajará entonces con la mitad de resolución o ¿por qué no? con una cuarta parte. Esto quiere decir que si se trabaja con la mitad de resolución, por cada punto del buffer se deben poner 2x2 pixels en pantalla; si se trabaja con una cuarta parte, por cada punto se pondrán 4x4 pixels, ¿de acuerdo? Vamos a elegir la mitad de resolución, o sea, 160x100.

Los accesos a la memoria de vídeo son más lentos que los accesos a la memoria convencional

Ahora aparece el primer problema de verdad del fuego, nuestro buffer será de 160x102 y se preguntará ¿por qué 102 y no 100?

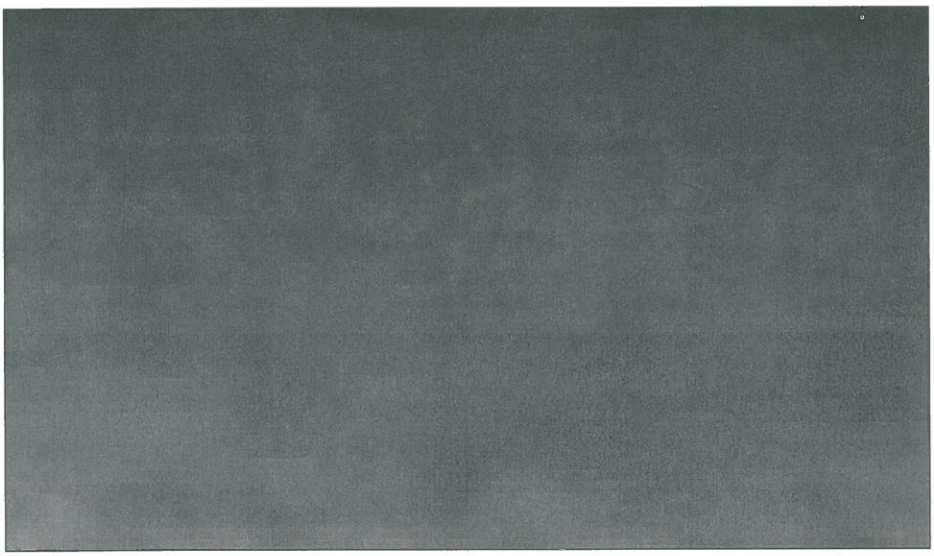
El creador de este efecto, el fuego, fue Javier Arévalo (Iguana) y según sus propias palabras "el fuego fue un bug que salió intentando hacer un plasma". Bien, cuando Javier programó su primer fuego, la última línea salía francamente fea, pero el resto era muy bonito. Esa línea será desplazada 2 líneas más hacia abajo. ¿Se entiende ahora por qué necesitamos dos líneas más?

Esto en lo que se refiere al buffer. Ahora se tratará la paleta. ¿Qué paleta se va a usar?. Lo que está claro cuando vemos un fuego es que emplea unos cuantos colores. Se pueden obtener fuegos bonitos desde 64 colores, pero usaremos los 256. Si lo que se quiere conseguir es "fuego" y no humo, agua u otro efecto similar, se debe crear una paleta que degrade de negro

a rojo en los 128 primeros colores, y de rojo a amarillo en el resto, la paleta que se incluye en el código será FIRE-PAL.INC

A continuación surge el segundo problema, ¿cómo calcular puntos aleatorios?

```
MOV   AX,[RandSeed]
MOV   BX,[RandSeed2]
MOV   BP,[RandSeed3]
ADD   AX,0a137h
ADD   BX,63f7h
ADD   BP,784Ah
ROL   AX,2
```



Como se puede suponer este efecto necesita algunos cálculos aleatorios, por lo que se va a crear una rutina que calcule puntos aleatorios entre 0 y 65535, para poder usarla en posteriores efectos:

```
; SEMILLAS ALEATORIAS
;
 RandSeed  DW 32B8h
 RandSeed2 DW 7F21h
 RandSeed3 DW 300Fh
; Número aleatorio 0 - 65535
; Out : AX
 GetRnd   PROC
    PUSH   BX BP
    MOV    [RandSeed],AX
    ADD    BX,AX
    ROR    BX,1
    MOV    [RandSeed2],BX
    SUB    BP,BX
    XOR    AX,BP
    MOV    [RandSeed3],BP
    ADD    AX,BX
    POP    BP BX
 GetRnd   ENDP
```

BUCLE PRINCIPAL

Por fin, acometemos el bucle principal de la rutina de fuego.

Escribimos la última línea del buffer con números aleatorios cuyo valor esté comprendido entre 128 y 255. Como el valor que devuelve GetRnd varía entre 0 y 65535, se hará:

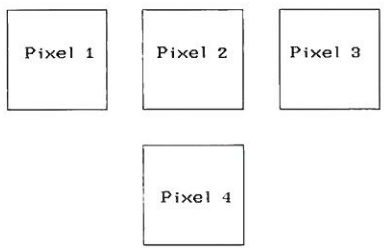
```
CALL   GetRnd
AND    AX,1111111b
ADD    AX,127
```

Y ahora se rellenará el resto del buffer. Y aquí es donde está el autentico "truquillo" del fuego: Para calcular cada pixel del buffer sumaremos los valores de los pixels que haya alrededor de este y calcularemos la media aritmética. Ese es el valor que buscamos para nuestro pixel. Se debe destacar en este apartado, que no es lo mismo rellenar el buffer comenzando por la parte escrita con valores aleatorios que comenzar a rellenar el buffer por la parte contraria. La diferencia se puede apreciar en el código incluido con este artículo: FIRE comienza a rellenar por la parte escrita, y OFIRE por la parte contraria.

Se esperará un retrazo vertical y se volcará el buffer de la memoria convencional a la de vídeo con la resolución elegida.

¿Sencillo verdad? En el próximo artículo veremos como se consigue el efecto de un túnel de estrellas. ■

$$\text{Valor del pixel} = \frac{\text{Pixel 1} + \text{Pixel 2} + \text{Pixel 3} + \text{Pixel 4}}{4} - K$$



SI ESTÁ ESPERANDO WINDOWS 95

Ya puede reservar su Microsoft® Windows 95, el gran acontecimiento del año y por tan sólo 16.900 ptas.* Resérvelo ahora mismo. Llámenos y podrá recibir gratis una maravillosa camiseta.

¡Disfrute de sus vacaciones y cuando vuelva Microsoft® Windows 95 le estará esperando!

Microsoft Windows 95 estará disponible a partir del 6 de septiembre de 1995.

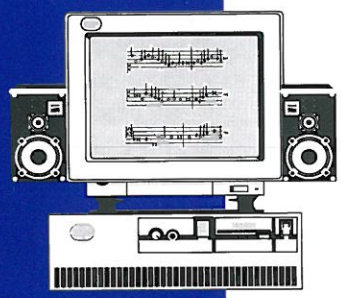
Inicio

Asegúrese ser el primero en recibirlo

(91) 804 00 96

* Transporte, seguro e IVA no incluído.

Microsoft
¿HASTA DONDE QUIERES LLEGAR HOY?



CÓMO REPRODUCIR UN FICHERO MOD

Luis Crespo

Tras una ausencia de varios meses, continuamos con la serie dedicada al sonido. En la entrega anterior (número 6 de la revista), se describía el formato de fichero musical MOD. En la entrega correspondiente al número 5, se explicaba cómo reproducir una digitalización utilizando un doble buffer, de forma que mientras suena medio buffer se va llenando el otro medio.

En este artículo se explica cómo, aplicando los conocimientos de las dos entregas anteriores, conseguir que nuestro programa haga sonar un MOD. Si bien hasta ahora todas las rutinas estaban escritas en C y Pascal, a partir de ahora algunas rutinas son críticas en cuanto a consumo de tiempo y se suministran en ensamblador. En cualquier caso esto no debe desanimar a nadie, ya que se explica en alto nivel la función que realiza cada rutina.

Ahora que ya sabemos reproducir una digitalización y el formato de los ficheros MOD, sólo falta interpretar su partitura mezclando los canales para hacerlos sonar.

MEZCLA DE CANALES

Ya sabemos reproducir una digitalización por cualquier dispositivo de sonido, ya sea una Sound Blaster o compatible, el altavoz interno o un DAC casero. Pero un módulo de sonido consta de 4 o más canales (puede llegar hasta 32), o lo que es lo mismo: pueden sonar 4 o más digitalizaciones simultáneamente. De modo que, como disponemos de un sólo DAC, necesitaremos emular 4 o más DACs con uno. Para conseguir que suenen varias digitalizaciones en un mismo instante, será necesario mezclarlas. La mezcla consiste simplemente en hacer en cada instante la media aritmética del dato que hay que enviar a cada DAC virtual. Es decir, para cada canal se extrae la muestra correspondiente al instrumento que está sonando, se suman todas ellas y se divide el resultado entre el número de canales: En este punto hay que tener en cuenta ciertos detalles: las muestras de los instrumentos de los MODs son datos de 8 bits. Cuando sumamos varios números de 8 bits nada nos asegura que el resultado vaya a caber en 8 bits. Por eso la suma de las muestras debe realizarse sobre un número de 16 bits. Por ejemplo, en el caso de 4 canales, al sumar 4 números de 8 bits en el peor de los casos necesitaremos un número de 10 bits para albergar el resultado, ya que 255*4=1020.

$$\text{Dato final} = \frac{\text{Dato Canal 1} + \ldots. \ldots. + \text{Dato Canal N}}{N}$$

La mezcla de canales se hace usando la media aritmética

Esto significa que, a pesar de que las muestras que se utilizan en los ficheros MOD son de 8 bits, el resultado de la mezcla generará datos de más bits, de forma que si se tiene una tarjeta de sonido de 16 bits debe aprovecharse esta calidad adicional.

Como dividir por un número indeterminado de voces es costoso, en la práctica se utiliza otro sistema: se modifica el volumen de cada canal. Esto se verá más claro a continuación.

VOLUMEN DE LOS CANALES

Como se vio en el artículo anterior, los instrumentos de un módulo de sonido tienen un campo dedicado al volu-

men por defecto. Además, existen comandos para modificar el volumen de una nota concreta. Para alterar el volumen de una digitalización podría utilizarse el mixer de la tarjeta Sound Blaster, pero necesitamos modificar los volúmenes de varios canales simultáneamente, cada uno con un volumen que puede ser distinto de los otros, así que de nuevo habrá que emular esta función por software.

La forma teórica de alterar el volumen de una digitalización consiste en dividir el valor de cada muestra por un factor. El resultado será una digitalización que varía entre rangos más pequeños, con lo que se disminuirá la amplitud de la señal, que es al fin y al cabo lo que nosotros percibimos como volumen. Por ejemplo, si dividimos cada muestra por 2, la amplitud de la onda disminuye a la mitad, y por lo tanto el volumen disminuye a la mitad. Aunque habría que entrar en consideraciones del tipo de que el oído humano percibe los cambios de volumen de forma logarítmica y no lineal, el parámetro de volumen de los módulos es lineal y por lo tanto podemos despreocuparnos de esta cuestión.

Sabiendo que el volumen varía entre 0 y 64, el factor que debería aplicarse para dividir cada muestra sería 64/volumen, ya que por ejemplo un valor de 32 significa "volumen a la mitad" y 64/32=2, un valor de 16 "volumen a la cuarta parte" y 64/16=4, etc. En lugar de dividir entre 64/volumen, podemos multiplicar por el volumen y dividir entre 64. El resultado será exactamente el mismo (una división es una multiplicación por el inverso), pero este sistema es mucho más práctico, ya que podemos postergar la división.

Esto es lo que se hace en realidad: simplemente se multiplica cada muestra por el volumen correspondiente, y el resultado se suma al resto de los canales. Al multiplicar un número de 8 bits por un valor que varía entre 0 y 64, el resultado, en el peor de los casos, necesitará un número de 14 bits: 255*64 = 16320, y 2^14 = 16384. En el caso de un MOD convencional en el que hay 4 canales, sumar 4 números de 14 bits en el peor de los casos ocupa 16 bits, por la misma regla.

Recordemos que habíamos postergado la división entre 64 (el volumen) y entre 4 (el número de canales). Pero dividir entre 64 y entre 4 es lo mismo que dividir entre 256, lo cual es lo mismo que desplazar 8 bits a la derecha, es decir, descartar los 8 bits bajos. Esto queda más claro con la siguiente expresión, en la que M1, M2, M3 y M4 son las muestras de cada canal:

$$\frac{M1*Vol1/64 + M2*Vol2/64 + M3*Vol3/64 * M4*Vol4/64}{4} =$$

$$\frac{(M1*Vol1 + M2*Vol2 + M3*Vol3 + M4*Vol4) / 64}{4} =$$

$$\frac{M1*Vol1 + M2*Vol2 + M3*Vol3 + M4*Vol4}{256}$$

En el caso de que el DAC sea de 16 bits (por ejemplo en el caso de una Sound Blaster 16), se pueden enviar directamente los 16 bits resultantes de la mezcla. El resultado es de mayor calidad que la de un sólo instrumento de un MOD; esto es lógico ya que durante la mezcla se ha aportado más información: los volúmenes y los canales simultáneos.

Volvamos ahora al caso general de N voces. Para 4 voces, el rango de volumen de 0 a 64 es ideal: no requiere de ninguna división al final de la mezcla y aprovechamos al máximo los 16 bits. Análogamente, en un módulo de 8 voces el rango ideal de volúmenes sería de 0 a 32, y en general el rango ideal será de 0 a 256 entre el número de voces. De modo que lo que se hará es adaptar el rango de volúmenes con los que trabaja el MOD al número de voces, y la mezcla de canales se verá reducida a una multiplicación y una suma por canal. De hecho, la rutina CalcVolume del fichero MODLOAD.PAS de la entrega anterior calcula precisamente eso: un volumen global en función del número de voces. Más adelante se verá cómo se implementa realmente la mezcla, pero todavía queda un punto a tener en cuenta: la frecuencia de muestreo.

FRECUENCIAS DE MUESTREO

Como se explicó en la entrega anterior, las distintas notas de un mismo instrumento se consiguen reproduciendo la digitalización correspondiente a distintas frecuencias de muestreo. Con una digitalización sólo sonando en cada instante, bastaría simplemente con reprogramar el dispositivo de sonido para que reprodujese el sonido a distintas frecuencias. Pero en cada canal de un módulo puede sonar simultáneamente una nota distinta, de modo que éste sistema no sirve. Como ya habréis adivinado, es necesario emular por software la posibilidad de reproducir al mismo tiempo distintos canales a distintas frecuencias de muestreo.

Partiendo de una frecuencia de muestreo fija e inamovible F, nos interesa reproducir una digitalización a cualquier otra frecuencia. Para reproducirla a una frecuencia 2*F bastaría con tomar una de cada dos muestras de la digitalización. De esta forma, en una unidad de tiempo se habrán recorrido el doble de muestras que a la frecuencia F, y si bien sólo se enviarán al DAC una de cada dos muestras, el efecto a nuestros oídos será el mismo que si se hubiesen enviado todas las muestras al doble de velocidad. Análogamente, para reproducir una digitalización a una frecuencia F/2 deberemos enviar al DAC dos veces cada muestra. Así, en una unidad de tiempo se habrá recorrido la mitad de muestras que a la frecuencia F.

Generalizando, necesitaremos un contador que nos indicará, qué muestra deberá enviarse al DAC en cada momento, y una variable de incremento que se sumará al contador en cada paso. Por ejemplo para una frecuencia 2*F el incremento valdrá 2. Pero el incremento puede tomar valores no enteros, como por ejemplo 1/2 para F/2 o 0.7324 o 3.1416. De forma que tanto el incremento como el

Hay que multiplicar cada muestra por el volumen del canal

contador deberán ser variables no enteras. El algoritmo sería el siguiente:

```
procedure RecorreDig(FrecDAC,FrecDig: Word)
 var
    Pos,Inc: Real;
 begin
  Inc := FrecDig/FrecDAC;
  Pos:=0;
  while Round(Pos)<TamanyoDig do
  begin
   MezclaMuestra(Digi[Round(Pos)]);
   Pos:=Pos+Inc;
  end;
 end;
```

Nótese que no es más que un algoritmo y no una rutina real, ya que se omiten cuestiones importantes. FrecDAC es la frecuencia fija del DAC y FrecDig es la frecuencia a la que debe reproducirse la digitalización. De todas formas, hay un problema en este ejemplo: se utiliza coma flotante, lo cual es muy ineficiente y muy complicado de implementar en ensamblador. Hay que tener en cuenta que la rutina MezclaMuestra se ejecutará F*N veces por segundo, donde F es la frecuencia de muestreo y N el número de canales, con lo que se puede ejecutar 176400 veces por segundo con 4 canales y 44100 Hz. Y para cada llamada habría que hacer una suma en coma flotante y un redondeo de coma flotante a entero.

El bucle principal de la mezcla puede ejecutarse más de 170.000 veces por segundo

Por ello, se emplea una técnica que es mucho más sencilla de implementar en ensamblador y al mismo tiempo consume mucho menos procesador que la coma flotante: los números en coma fija.

LA COMA FIJA

Si bien no es el momento de extenderse en el tema, una pequeña explicación aclarará en qué consiste esta técnica. Un número en coma fija se guarda en memoria y se opera como un número entero multiplicado por una constante, es decir, de la forma A*K. La suma de dos números en coma fija es directa: A*K + B*K = (A+B)*K, que es otro número en coma fija. Se trabaja internamente en coma fija y en el momento en el que se necesita pasar de coma fija a entero basta con dividir entre K. En el recuadro 1 se ilustra un ejemplo con K=100, lo cual da dos decimales de precisión.

RECUADRO 1

En punto flotante, 3.7 + 2.14 = 5.84

En punto fijo:
3.7*100 = 370 ; 2.14*100 = 214
370 + 214 = 584
Paso de punto fijo a punto flotante: 584/100 = 5.84
Paso de punto fijo a entero, truncando la parte fraccionaria: 584/100 = 5

Ejemplo de aritmética de punto fijo.

Tomando la constante K=256, además de disponer de la suficiente precisión, se tiene otra ventaja: en binario, multiplicar por 256 es desplazar 8 bits a la izquierda y dividir por 256 es desplazar 8 bits a la derecha. Esto facilita mucho las operaciones, ya que simplemente se descartará el byte bajo de un número en coma fija para truncarlo a entero.

Ahora el algoritmo anterior se escribiría en coma fija de la siguiente manera:

```
procedure RecorreDig(FrecDAC,FrecDig: Word)
var
  Pos,Inc: LongInt;
begin
  Inc := (FrecDig SHL 8) DIV FrecDAC;
  Pos:=0;
  while Pos SHR 8 < TamanyoDig do
  begin
    MezclaMuestra(Digi[Pos SHR 8]);
    Pos:=Pos+Inc;
  end
end;
```

CÁLCULO DEL INCREMENTO

Pero la cosa no termina ahí, porque en cada nota de un MOD no viene especificada a qué frecuencia deberá reproducirse cada digitalización, sino que el tono de la nota se determina mediante un valor proporcional al inverso de la frecuencia: el período. Por cuestiones propias del hardware del Amiga, la frecuencia de muestreo se calcula dividiendo el "número mágico" 3579545.25 entre el valor del período. De modo que el incremento se calcula en función del período de la siguiente manera:

FrecDig = 3579545.25/Periodo
Inc = (FrecDig*256)/FrecDAC = (256 * 3579549.25/Periodo) / FrecDAC

Todos estos valores excepto el período se conocen de antemano, de forma que se pueden precalcular y almacenar en una variable, que en las librerías suministradas se llama PeriodInc:

PeriodInc := 256*3579545.25 / FrecDAC

Y en el momento de calcular el incremento para cada nota, bastará con dividir PeriodInc por el período correspondiente:

Inc := PeriodInc/Period

DACS VIRTUALES

Como se ha visto, la reproducción de un MOD requiere emular por software varios DACs o canales independientes, cada uno con su propia digitalización, su propio volumen y su propia frecuencia de muestreo. Por eso nues-

RECUADRO 2

```
TYPE
  TChannelInfo=RECORD
    Playing,
    Loop:Boolean;
    Step:LongInt;
    Volume:Byte;
    pSample:Pointer;
    LoopOfs,
    EndOfs,
    PosOfs:Word;
    PosFrac:Byte;
  END;

VAR
  Channel:ARRAY[1..MaxChn] OF TChannelInfo;
```

Estructura de datos de DAC virtual.

tro programa necesitará una estructura de datos para cada DAC que emule, en la que se controle el volumen, la posición y el incremento entre otras cosas. De eso se encarga la estructura de datos TChannelInfo, que se muestra en el recuadro 2.

RECUADRO 3

```
Contenido de los registros:

  DH: 8 bits de menor peso del índice de la
muestra en punto fijo
  BX: 16 bits de mayor peso del índice de la
muestra en la digitalización

    Es decir: el índice en punto fijo está forma-
do por los registros
    BH-BL-DH, con los bytes de mayor peso a
la izquierda.

  CX: Incremento en punto fijo
  DL: Volumen del canal
  SI: Offset del buffer a llenar
  ES: Segmento del instrumento
  DI: Tamaño de la digitalización
  BP: Contador del bucle: tamaño del buffer a
llenar


 @Otro:
   CMP BX,DI              ; Miro si he de
salir antes porque se...
   JAE @SalAntes          ; ...ha llegado al
final del instrumento
 @Mas:
   MOV AL,[ES:BX]         ; Leo muestra
del instrumento
   IMUL DL                ;
Multiplico por el volumen
   ADD [SI],AX            ; Coloco el
resultado en el buffer
   @Sigue:
   ADD SI,2               ;
Paso al siguiente dato del buffer
   ADD DH,CL              ; Sumo partes
fraccionarias
   ADC BL,CH              ; Sumo partes
enteras
   ADC BH,0
   DEC BP                 ;
Decremento contador del bucle
  JNZ @Otro               ; y
repito
```

Bucle principal de la rutina de mezcla.

El campo Playing es un boolean que indica si ese canal está sonando en ese momento o no. En caso de que no suene directamente se ignora, lo cual ahorra bastante tiempo.

Loop es otro boolean que indica si el instrumento que se está reproduciendo tiene o no loop (ver artículo anterior). De esta forma, si durante la mezcla se llega al final de la digitalización, se volverá al inicio de loop.

Step es el incremento en aritmética de coma fija, es decir, la variable Inc del apartado anterior.

Volume es el volumen del canal: el número por el que hay que multiplicar cada muestra antes de sumarla al buffer.

La coma fija consume mucho menos procesador que el punto flotante

pSample es el puntero a la digitalización.

LoopOfs y EndOfs indican la posición de inicio de loop y final de instrumento, respectivamente.

PosOfs y PosFrac, juntas, forman un número de 3 bytes en coma fija, en el que PosOfs alberga la parte entera y PosFrac la parte fraccionaria de la posición de la muestra actual en la digitalización.

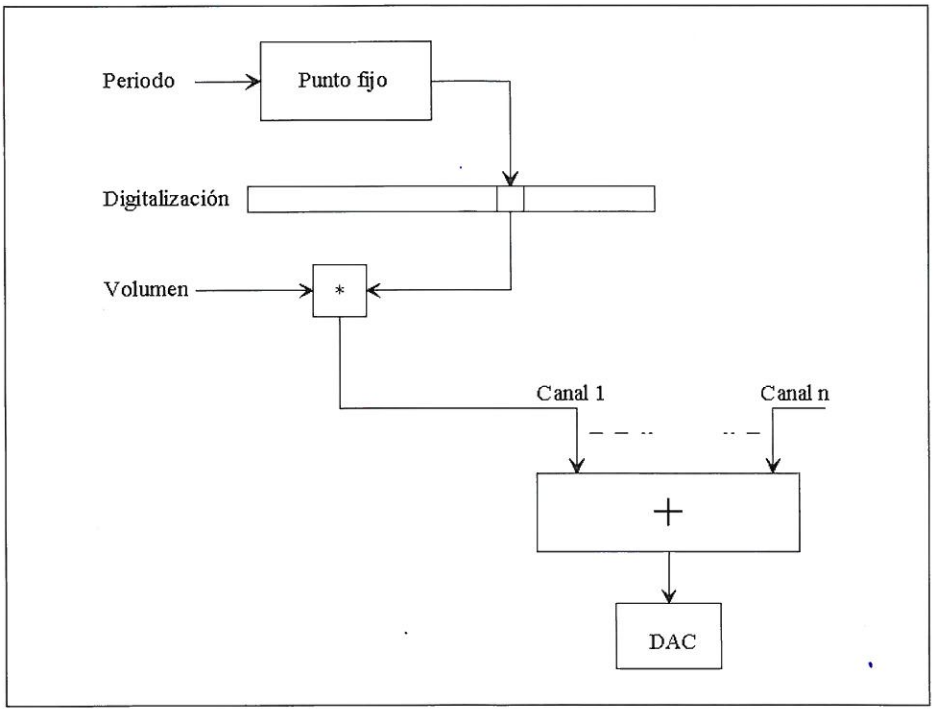


Figura 1: Esquema del proceso de mezcla de canales.

Repasemos el proceso completo de la emulación de varios canales con un sólo DAC: para cada canal se recorre el instrumento activo con un incremento en coma fija y se extrae la muestra correspondiente. Dicha muestra se multiplica por el volumen del canal y el resultado se suma junto con los resultados del mismo proceso en otros canales. El número resultante se envía al DAC, ya sean los 16 bits o los 8 bits altos. Un esquema del proceso puede verse en la figura 1.

Para generar la música, el programa está continuamente mezclando tantos canales como sea necesario. Por otra parte una rutina se encarga de determinar qué instrumento debe sonar en cada instante por cada canal, a qué volumen y a qué frecuencia, y consecuentemente modifica el contenido de los canales. Con esto se consigue que se vayan sucediendo las notas y los instrumentos, obteniéndose la melodía.

IMPLEMENTACIÓN

Es el momento de implementar el proceso de mezcla que se ha explicado en los párrafos anteriores. El sistema más inmediato que se nos puede ocurrir es programar el controlador de interrupciones para que genere tantas interrupciones por segundo como la frecuencia de muestreo (al estilo del primer artículo de la serie), y en cada interrupción mezclar una muestra de cada canal y enviar el resultado de la suma al DAC. Pero cada vez que se entra

en una rutina de interrupción hay que salvar todos los registros que se van a utilizar, cargarlos con los valores que nos interesan, ejecutar la operación y restaurarlos antes de volver. Esto supone una carga de procesador muy grande.

El tamaño ideal del buffer es de F/50 muestras

En la segunda entrega de la serie se explicó cómo podía utilizarse el controlador de DMA para que de forma automática fuese transfiriendo muestras a la velocidad necesaria. También se vio que se podía utilizar un buffer de doble tamaño, y mientras el DMA se encargaba de transferir una mitad, el procesador se encargaba de llenar la otra. Este sistema es mucho más eficiente, ya que permite que la rutina de mezclado trabaje con todos sus datos en los registros del procesador, y por otra parte se producen muchas menos interrupciones por segundo.

La rutina NextTick se encarga de la mezcla de canales y decodificación de la partitura

El sistema es el siguiente: supongamos que el tamaño del doble buffer es de 2*N muestras de 16 bits cada una (recuérdese que la mezcla debe realizarse sobre un número de 16 bits para no perder precisión), luego una mitad del doble buffer tendrá N muestras. Se empieza llenando el buffer con ceros. Para cada canal, se extrae la muestra correspondiente según indique el contador en coma fija, se multiplica por el volumen y se suma al valor en la posición actual del buffer. Se repite el proceso N veces, una para cada muestra del buffer, de modo que tras recorrer todos los canales, en el buffer hay N muestras resultado de la mezcla, listas para ser enviadas al DAC. El algoritmo sería el siguiente:

```
for i:=0 to N do Buffer[i]:=0;
{Limpia buffer}
  for i:=1 to NumCanales do
    for j:=1 to N do
      Buffer[j] := Buffer[j] +
ObtenMuestra(i,j);
```

En donde la rutina ObtenMuestra(i,j) obtendría la muestra j del canal i, ya multiplicada por el volumen. Esta rutina será llamada cada vez que se termine de reproducir una mitad del doble buffer, para seguir mezclando canales.

Si el dispositivo de sonido permite reproducción de digitalizaciones de 16 bits, el buffer que se crea puede enviarse directamente al DAC. En caso de que el DAC funcione a 8 bits, será necesario tomar el byte alto de cada word y copiarlo en el buffer definitivo. Pero hay otro detalle a tener en cuenta: los instrumentos de los MODs utilizan muestras con signo, es decir, sus valores oscilan de -128 a +127. El DAC de la Sound Blaster, en cambio, utiliza muestras sin signo, de forma que hay que sumar 128 a cada byte para que sus valores oscilen entre 0 y 255. Como interesa que las rutinas sean lo bastante genéricas como para poder soportar tanto 8 como 16 bits, se utilizan dos buffers: uno en el que se va mezclando el sonido y otro en el que se copia o adapta el sonido ya mezclado y se envía por DMA a la tarjeta de sonido.

Para entender mejor todo el proceso, lo mejor es mirar a conciencia el recuadro 3 en el que se lista el bucle principal de la rutina de mezcla en ensamblador, e identificar la función de cada instrucción.

¿Cuál es el tamaño ideal para los buffers? Recordemos que la unidad básica de tiempos de los MOD era el cincuentavo de segundo. Por lo tanto, un buffer que se tuviese que actualizar 50 veces por segundo es ideal, porque justo al entrar en la interrupción se puede mirar si es el momento de cambiar de nota. Si a una frecuencia de muestreo F se envían F muestras por segundo al DAC, haciendo el buffer de F/50 muestras de tamaño se nos generarán 50 interrupciones por segundo (como utilizamos un doble buffer, el tamaño total del buffer será del doble de muestras, pero las rutinas de mezcla sólo llenarán una de las mitades en cada interrupción).

EL PROGRAMA DE EJEMPLO

El programa MODPLAY.EXE es una primera versión de un reproductor de ficheros MOD. MODPLAY carga el MOD especificado como parámetro (debe incluirse la extensión) y lo reproduce mientras muestra sus datos por pantalla. Las teclas + y - avanzan y retroceden un patrón respectivamente, y la tecla ESC abandona el programa.

Al fichero SONGS.PAS se le han añadido dos nuevos procedimientos: SongPlay y SongStop, que se encargan de iniciar y terminar la reproducción del módulo. SongPlay pone en marcha la reproducción del doble buffer. La rutina que debe ejecutarse en cada interrupción se encarga de llamar a la que mezcla los canales y decodifica la partitura. Desde la interrupción también se actualizan los punteros a los buffers y se adapta el buffer de 16 bits a otro de 8 bits que usa la rutina de reproducción, ya que de momento sólo está implementada la reproducción de digitalizaciones de 8 bits y mono.

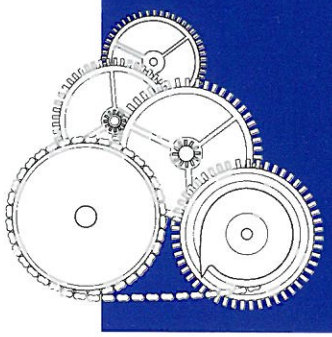
El nuevo fichero PLAY.PAS contiene las rutinas de mezcla de canales y decodificación de partitura. Además se implementan ya los comandos explicados en la entrega anterior: volumen, cambio de tempo y saltos en la partitura, aunque de momento se ignoran los cambios de BPM. La rutina NextTick es la encargada de controlar todo esto, y es llamada tras recorrer cada mitad del doble buffer.

EN LA PRÓXIMA ENTREGA

Todavía quedan algunos puntos por aclarar, como por ejemplo cómo se sabe qué nota debe sonar en cada momento o cuándo debe avanzarse a la siguiente nota en la partitura. Además de esto, en la siguiente entrega de la serie se explicará a fondo el funcionamiento de los efectos. ■

ADQUISICIÓN DEL CONOCIMIENTO

Juan José Samper



Una cantidad adecuada de conocimientos es imprescindible para reproducir el comportamiento intelectual humano. Es necesario identificar, dar forma, y representar los conocimientos, a veces bastante simples, que permiten razonar. Lo ideal es que el sistema experto haga todo eso por sí solo.

Hasta en los trabajos más sencillos, el ser humano recurre a multitud de conocimientos de todo tipo, pero sobre todo heurísticos: reglas prácticas e imprecisas, aunque casi siempre eficaces. El problema es que, normalmente, esos conocimientos no se expresan. La mayoría no están escritos en ningún libro, y sólo se pueden aprender con la práctica. Otro problema se plantea debido a que es difícil conocer anticipadamente la cantidad de conocimientos que necesita un sistema experto para efectuar su tarea correctamente. Todo ello hace que la adquisición del conocimiento sea una necesidad fundamental además de una operación económicamente importante.

FUNDAMENTOS DE LA INGENIERÍA DEL CONOCIMIENTO

El término "ingeniería del conocimiento" engloba a los científicos, tecnología y metodología necesarios para procesar el conocimiento. Su objetivo es extraer, articular y computerizar el conocimiento de un experto.

En la ingeniería del conocimiento hay tres conceptos interrelacionados: dato e información, propios de la informática, y conocimiento, que es propio de la informática junto a la inteligencia artificial.

El universo de discurso es el conjunto de objetos relativos a un problema o situación a considerar.

Un dato es la representación simbólica de un aspecto simple, objeto o hecho, del universo de discurso. Se distingue entre datos objeto, que son elementos de determinado dominio (pueden ser pares atributo-valor) y datos hecho, que son relaciones entre objetos.

Información es cualquier estímulo capaz de cambiar el estado o de alterar la conducta de un sistema. Es todo aquello que aumenta el conocimiento, o sea, que es novedoso. La cantidad de información que se obtiene de un mensaje depende de cuánto reduzca la incertidumbre en el receptor acerca del correspondiente universo de discurso.

El conocimiento es la representación simbólica de aspectos complejos de algún universo de discurso. Puede ser de dos tipos:

- Declarativo: describe cómo son las cosas.
- Procedural: describe cómo funcionan las cosas.

El uso de un tipo o de otro depende del entorno, aunque en los sistemas expertos se suele usar el conocimiento declarativo por ser más cómodo y fácil de representar.

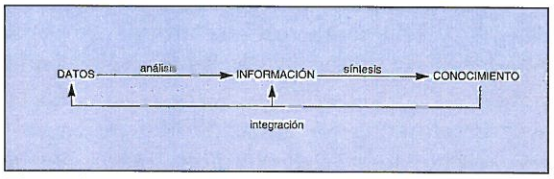


Figura 1.

El proceso de formación del conocimiento de una persona normalmente es cíclico, y tiene el esquema que se muestra en la figura 1.

El conocimiento llega incompleto y desordenado. El ingeniero del conocimiento es la persona encargada de organizarlo, completarlo y prepararlo para su transferencia al sistema.

El metaconocimiento es conocimiento sobre el propio conocimiento. Cuanto peor uniformizada esté una tarea, más metaconocimiento se necesitará para llevarla a cabo.

La habilidad es la capacidad para

Una exquisita facultad de razonamiento no compensa nunca la ignorancia. Gracias al aprendizaje automático, un sistema experto puede aprender por sí mismo observando a un especialista humano en acción.

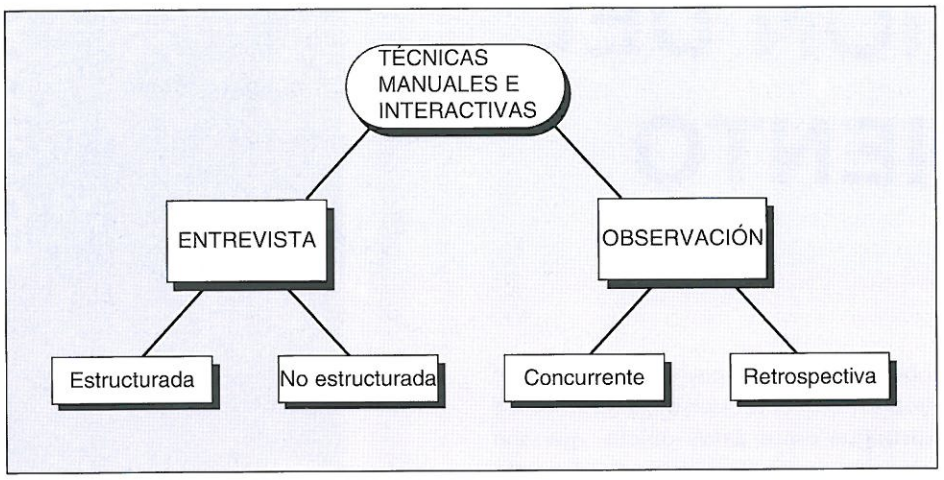


Figura 2.

trabajar con conocimiento parcial, por parte del experto, y obtener una alta productividad. Es deseable una tarea hábil a una bien informada pero burda. Lo que interesa es reproducir y construir sistemas hábiles. Un sistema no es un sistema experto si no es hábil.

NIVELES DEL CONOCIMIENTO

Cuando el experto ha terminado de relatar el conocimiento que posee, se puede establecer la base de conocimiento de orden 0 (o teoría de orden 0), formada por conocimiento factual: es el que se puede conocer sin necesidad de hacer preguntas. Si se monta un sistema experto con este conocimiento, y se intenta resolver un problema, el resultado va ser bastante pobre.

Normalmente, los expertos humanos son capaces de dar un paso más, proporcionando una heurística (o teoría de orden 1), que se puede definir como el conocimiento que tiene el especialista a partir de sus propias experiencias. La heurística son reglas que muestran cómo tratar el conocimiento. Es un conocimiento sobre el propio dominio en el que se trabaja, que dice cómo emplear los conocimientos que el experto humano dio como factuales. Añadiendo la heurística a la base de conocimientos de un sistema experto se mejora la respuesta, pero se comprueba que no se llega al grado de experto que tiene el especialista humano.

Volviendo al experto, se obtiene el metaconocimiento (o teoría de orden 2), que es independiente del dominio y dice cómo emplear eficientemente el conocimiento. Es la parte del conocimiento que es completamente independiente del contexto. Dentro del metaconocimiento habrá reglas que indican cómo actuar fuera de cualquier dominio. Puede servir para detectar fallos sintácticos y semánticos en la transmisión del conocimiento por parte del experto. Incluso la forma en la que el experto representa el conocimiento es metaconocimiento. El metaconocimiento es algo innato en los expertos, por lo que difícilmente hablan de el.

Para obtener metaconocimiento, se examina el conocimiento que ha proporcionado el especialista humano. Cuando el sistema no responde de forma adecuada, se analizan los fallos y se descubren las metareglas (reglas de metaconocimiento) que se deberían incluir.

UTILIDAD DEL METACONOCIMIENTO

- Ayuda a que el sistema se adapte al entorno.
- Permite resolver conflictos entre reglas.

Un ejemplo son las metareglas siguientes:

M.R.1: Usar reglas que empleen materiales ligeros antes que otras con materiales pesados.

M.R.2: Usar reglas proporcionadas por expertos veteranos mejor que por expertos más novatos.

- Permite detectar y suplir fallos simples en las reglas.

Ejemplo:

M.R. sintáctica: Toda regla debe tener condición y acción.

M.R. semántica: Toda regla debe tener por lo menos un caso sobre el que aplicarla.

ADQUISICIÓN DEL CONOCIMIENTO

La adquisición de conocimiento es la transferencia de conocimiento desde alguna fuente a la base de conocimientos del sistema. Esto implica aplicar alguna técnica para extraer el conocimiento del especialista, interpretar los datos para reconocer el conocimiento fundamental, y construir un modelo de representación (como, por ejemplo, reglas del tipo SI...ENTONCES...).

Las fuentes del conocimiento son los expertos humanos, libros, y datos experimentales agrupados en bases de datos.

La adquisición del conocimiento y su modelización son las tareas que más tiempo requieren, y que más dificultades presentan al construir un sistema experto. Puede ocurrir que no haya expertos, que no sean tan expertos, que los expertos no quieran dar su conocimiento o no puedan (porque sea intuitivo). No es fácil encontrar los trucos del experto, el cómo aborda la falta de información. El conocimiento más difícil de extraer es el que el experto lleva innato.

Como resultado de la adquisición, se construye una base de conocimientos. En el caso de reglas de producción, se

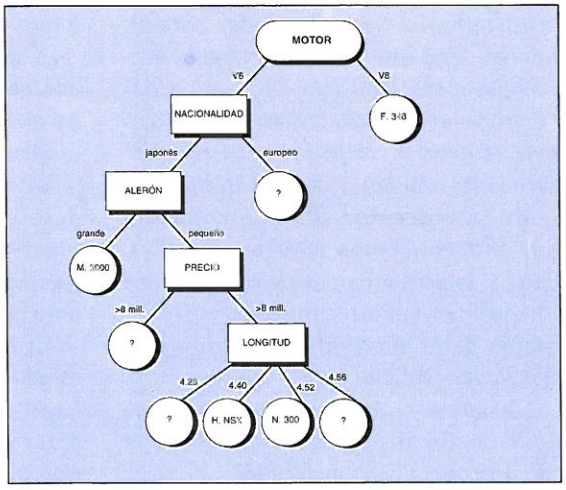


Figura 3.

construyen nuevas reglas y se incorporan al sistema.

La adición de conocimiento puede ser directa, o como resultado de una inducción a partir de ejemplos, o como deducción de otros conocimientos ya almacenados en la base de conocimientos. La inducción es el proceso de producir principios generales a partir de ejemplos específicos, mientras que la deducción significa extraer conclusiones específicas a partir de principios generales.

La adquisición del conocimiento tiene tres fases:

1ª. Sacar datos del dominio que interesa.

2ª. Elaborar los datos para obtener información.

3ª. Codificar el conocimiento de alguna forma.

Ejemplo: Sistema con una "caja negra", del que sólo se puede conocer su entrada x y su salida y. Lo que se quiere saber son las reglas que sigue.

Se observan los siguientes datos:

Lingüísticamente, puede decirse que dada una entrada, el sistema aproxima la misma salida. Entonces, este sistema tiene la ecuación siguiente, donde r es el "ruido":

Esto es ya una representación simbólica de conocimiento.

TÉCNICAS DE ADQUISICIÓN DEL CONOCIMIENTO

En general, las técnicas más representativas para la extracción del conocimiento se dividen en manuales, y basadas en el ordenador. Las más simples se basan en un contacto directo entre el ingeniero del conocimiento, y un experto en el dominio, sin usar herramientas especializadas. También pueden ser por contacto entre el ingeniero del conocimiento y los datos. Las técnicas manuales consumen mucho tiempo y son costosas. Para simplificar el proceso, se han construido una variedad de técnicas asistidas por ordenador, en las que se pone en contacto la máquina con la fuente de conocimiento. El ordenador pregunta al especialista al igual que lo haría un ser humano (en este sentido no se diferencia de lo que haría un sistema manual). Pueden ser interactivas (o semiautomáticas) y de aprendizaje automático.

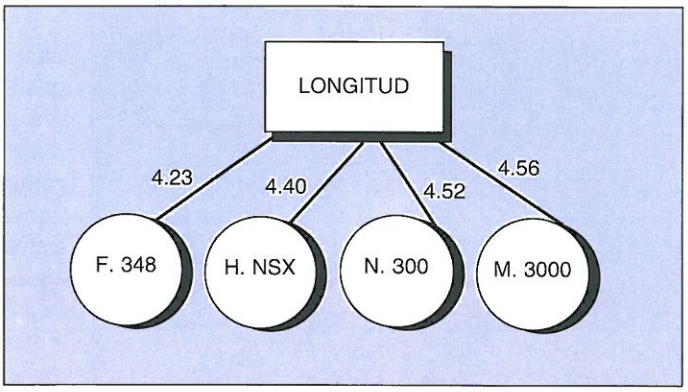


Figura 5.

En muchos casos, para llegar a la heurística y deducir el metaconocimiento, se emplean técnicas psicológicas. Estas técnicas se usan sobre todo en grandes empresas, cuando el experto no es muy colaborador.

TÉCNICAS MANUALES E INTERACTIVAS

El contacto entre un experto en el dominio y un ingeniero del conocimiento se hace directamente, de persona a persona, o con la ayuda de algún software especial. En la figura 2 se muestra un esquema de las técnicas manuales e interactivas. Las principales son la entrevista con el experto y la observación del experto.

La entrevista puede ser:

• Estructurada: El entrevistador o ingeniero del conocimiento tiene un plan bien programado, cada respuesta le permite sacar una información, y va construyendo la base de conocimientos. Avanza de forma estructurada hacia la obtención del conocimiento.

• No estructurada: Se deja llevar por la conversación. Se habla sin orden y sólo al final se puede construir la base de conocimientos.

Se suele empezar con entrevistas no estructuradas para alcanzar una idea general, y luego se pasa a realizar entrevistas estructuradas para afinar en detalles preguntando por cuestiones más específicas.

Para efectuar las entrevistas se debe:

1. Seleccionar los mejores expertos en el tema.
2. Tomar contacto tanto con el experto como con el dominio de su trabajo.
3. Organizar y conducir las entrevistas. Es importante generar documentación sobre las entrevistas y construir la base de conocimientos desde el principio, teniendo cuidado con la incertidumbre.

El ingeniero de conocimiento emplea la mayor parte de su tiempo en preparar las sesiones y en analizarlas. Anotará las metas logradas en la entrevista y una descripción de los datos extraídos. Posteriormente, elaborará las reglas derivadas de la sesión (suponiendo un sistema experto basado en reglas). El proceso de la entrevista se considera más un arte que una ciencia.

Cuando se entrevista a varios expertos, se realiza un debate entre ellos (también llamado brainstorming o torbellino de ideas), una consulta nominal (los expertos no se ponen de acuerdo y votan) y, por último, se intenta forzar un consenso. No es fácil llegar a una decisión consensuada. Si no coinciden los distintos expertos, se

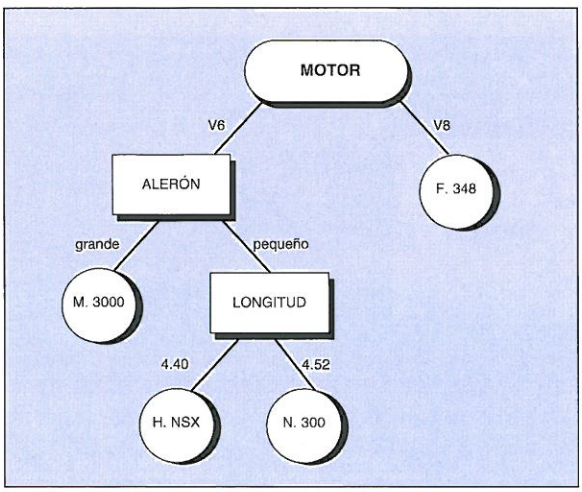


Figura 4.

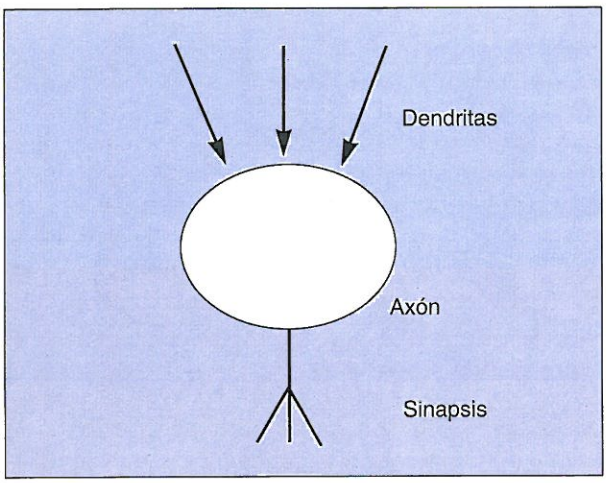


Figura 6.

le da un valor al conocimiento de cada uno y se introduce en la base de conocimientos por separado. Para resolver conflictos, se coge el metaconocimiento que provenga del experto con más experiencia.

La observación es una alternativa o un complemento a la entrevista, ya que muchas veces el experto no puede expresar sus conocimientos específicamente o puede tener dificultades para hacerlo. En estos casos es más provechoso adquirir el conocimiento mediante observación: se trata de observar el comportamiento del especialista humano ante un problema y extraer las leyes que lo rigen. La observación puede ser concurrente, si el ingeniero de conocimiento y el experto concurren en la actuación, o retrospectiva, cuando se graba al experto y esta grabación se le presenta al experto para que explique qué es lo que hace.

Normalmente, para complementar la entrevista, el ingeniero de conocimiento realiza una observación de lo que le ha dicho el experto.

APRENDIZAJE AUTOMÁTICO

El aprendizaje automático ya interesaba a los antiguos filósofos. Uno de los principales logros de la inteligencia artificial actual es que puede otorgar a los ordenadores la habilidad de aprender.

El objetivo es transferir al interior de la máquina el conocimiento que haya en su exterior. El aprendizaje puede ser:

- Memorístico, en el que se transcribe el conocimiento, se copia la memoria de un ordenador a otro.
- Por relato, que se basa en adaptar el conocimiento desde su fuente a una forma que pueda ser aceptada y utilizada por el ordenador. "Se le cuenta" el conocimiento. A veces se transfiere sólo una parte y se reutiliza de forma distinta a como fue concebido.
- Por analogía, si se guarda el conocimiento que resuelve un problema para aplicarlo cuando las circunstancias sean similares.
- Inductivo, al ordenador se le suministran datos y de ellos extrae el conocimiento. Se efectuará:
- A partir de ejemplos (supervisado o con maestro), hay un ser inteligente que le indica a la máquina cuáles son los ejemplos, y una vez conocidos, ésta ya puede trabajar y sacar conclusiones, aprendiendo e induciendo un concepto de más alto nivel. Se le suministran asociaciones entre objetos, y obtiene la regla resultante. Ejemplo, si se le dan los pares (1, 1), (2, 2), (3, 3), se induce x = y.
- A partir de observaciones (no supervisado o sin maestro), no hay un ser inteligente, el ordenador sólo "se pone a observar" y extrae conocimiento, descubre regularidades de un grupo de datos.

Los paradigmas más utilizados para el aprendizaje automático son:

- Inducción de reglas a partir de bases de datos
- Razonamiento basado en casos (CBR)
- Redes neuronales
- Algoritmos genéticos

APRENDIZAJE A PARTIR DE BASES DE DATOS

Entre los métodos de aprendizaje automático, el aprendizaje a partir de bases de datos por inducción (ya sea mediante ejemplos u observación), es prácticamente el único que se utiliza en los sistemas expertos. Este aprendizaje está particularmente adaptado para resolver problemas de clasificación, lo cual no es restrictivo, porque la mayoría de los problemas se pueden enfocar como uno de ese tipo.

La técnica se desarrolla en los siguientes pasos:

1º .Identificar objetos, atributos y valores.
2º .Construir el árbol o la tabla de decisión.
3º .Construir la base de conocimientos.

Ejemplo: Problema de clasificación de coches, en el que se consideran sólo cuatro modelos: Ferrari 348, Honda NSX, Nissan 300 y Mitsubishi 3000.

Identificación de objetos, atributos y valores:

Los objetos son los cuatro tipos de coches. Habrá que determinar qué atributos sirven para diferenciar los coches entre sí, descartando aquellos que no se necesiten para el proceso de toma de decisiones. Por ejemplo, el atributo "categoría de coche" no es necesario, puesto que, todos los coches considerados son deportivos. Se tendrá cuidado en que los atributos desechados no formen parte del co-

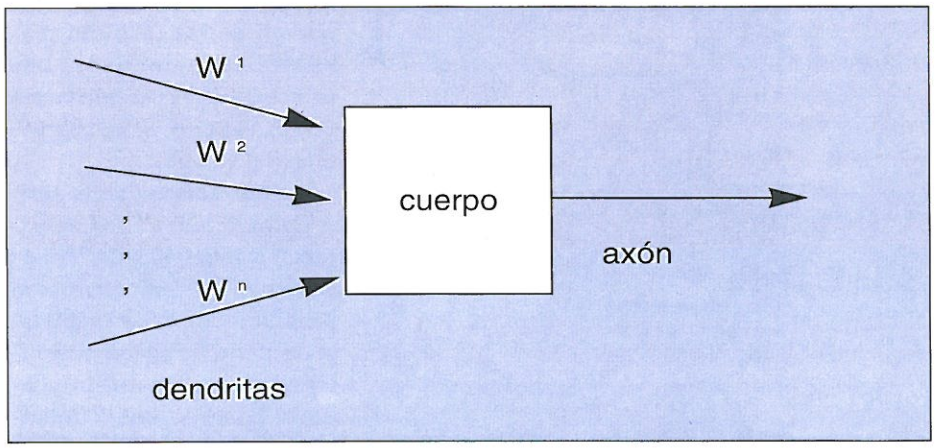


Figura 7.

nocimiento necesario para resolver el problema. Una vez elegido el conjunto de atributos a utilizar, se confecciona una lista con ellos y con sus valores asociados, formando la tabla 1.

ESTABLECIMIENTO DEL ÁRBOL DE DECISIÓN

Cada nodo en el árbol representará o una pregunta sobre el valor del atributo o una conclusión. Cada rama que emana de un nodo perteneciente a una pregunta, representará uno de los posibles valores del atributo asociado. Los nodos correspondientes a preguntas se representarán mediante cajas, mientras que aquellos que reflejen conclusiones se representarán mediante círculos. Usando de forma arbitraria el atributo "tipo de motor" como nodo raíz, se obtiene el árbol de la figura 3.

Se puede desarrollar un árbol de decisión alternativo, como se muestra en la figura 4, simplemente ordenando los nodos y eliminando cualquier rama que lleve a conclusiones imposibles. Este árbol requerirá menos atributos para llegar a una identificación, y nunca resultará en un caso imposible. En general, se preferirá desarrollar un árbol con tan pocos atributos como sea posible, y con todos los casos representando conclusiones conocidas. Cuanto menor sea el número de atributos, menos datos y/o respuestas de usuario se necesitan para llegar a una conclusión.

GENERACIÓN DE LAS REGLAS A PARTIR DEL ÁRBOL

Una cadena se define como el camino desde un nodo del árbol a otro, recorriendo las ramas en una sola dirección.

Para generar las reglas, primero se identifica un nodo conclusión que no haya sido tratado, se traza una cadena, a través del árbol, desde el nodo conclusión hasta la raíz, y, por último, se construye la regla correspondiente a la cadena considerada, repitiendo este proceso para cada nodo conclusión. Los nodos círculo formarán la parte conclusión de la regla (parte ENTONCES...), mientras que los nodos rectángulo formarán la parte condición o premisas de la regla (parte SI...).

Aplicando el algoritmo al árbol de la figura 4, que no contiene ningún nodo indeterminado, se obtienen las siguientes reglas:

Esta es ya una base de conocimientos para el problema de identificación de coches. Sin embargo, el conjunto de reglas es poco eficiente, considerando por eficiente el que tenga pocas condiciones. En realidad, sólo se necesita preguntar por la longitud del coche para discernir de cuál de ellos se trata. El atributo más importante para este ejemplo (el que contiene más información) será "longitud". El problema se simplifica mucho, obteniéndose el árbol minimizado de la figura 5. De él, se genera el siguiente conjunto de reglas:

Sin embargo, puede resultar imprudente minimizar el número de atributos, ya que, una base de reglas puede ser extensivamente modificada en un futuro, requiriendo que se incluyan atributos no necesarios previamente.

En el ejemplo, se han desarrollado árboles diferentes de forma arbitraria, dependiendo del orden de selección de los atributos (particularmente de la elección del nodo raíz). En un problema real, se debe de utilizar un enfoque sistemático para desarrollar un conjunto de reglas eficiente. Un mecanismo que se utiliza para tratar este problema es el algoritmo ID3 de J.R.Quinlan, que calcula qué atributos se deben seleccionar primero.

Tipo de coche				
Atributo	Ferrari 348	Honda NSX	Nissan 300	Mitsubishi 3000
Tipo de motor	V8	V6	V6	V6
Alerón	Pequeño	Pequeño	Pequeño	Grande
Nacionalidad	Europeo	Japonés	Japonés	Japonés
Precio	> 8 mill.	< 8 mill.	< 8 mill.	> 8 mill.
Longitud (m)	4.23	4.40	4.52	4.56

Tabla 1. Lista de atributos de coches.

RAZONAMIENTO BASADO EN CASOS

Este tipo de razonamiento utiliza la experiencia que se adquirió al tratar casos o problemas similares. Es un aprendizaje por analogía.

Se construye una base de conocimiento con los casos que ya se han resuelto. Cuando aparece un problema nuevo, se buscan problemas similares en la base de casos, y se adaptan sus soluciones a la situación actual. Un sistema experto de esta clase va a proporcionar la mejor solución disponible, más que una solución exacta.

Si con los casos existentes en la base no se puede resolver una situación de forma satisfactoria, ésta se atrapa automáticamente para actualizar los casos disponibles. Al irse incorporando nuevos casos, el sistema irá aumentando sus prestaciones.

REDES NEURONALES

Se trata de emular las neuronas cerebrales mediante modelos matemáticos, usando unidades de procesamiento elementales que copian algunas de sus características.

Una neurona biológica tiene el esquema que se muestra en la figura 6. Cada axón se conecta a las dendritas de otras células, formando redes capaces de transmitir impulsos eléctricos.

El modelo más simple es el diseñado por Mc Culloch y Pitts en 1949, en el que una caja representa el cuerpo de las neuronas. Tiene entradas que son las dendritas, las cuales están valoradas con un peso (w) que representa la potencia de la conexión. La salida representa el axón. Este modelo es el de la figura 7. Los pesos de las dendritas pueden ser positivos o negativos. En el primer caso se tendrán canales de excitación y en el segundo canales inhibitorios. El cuerpo procesa las señales que le llegan de otras neuronas según la hipótesis siguiente:

- Señales de entrada: Oi = 0 o 1.
  Se ponderan según su peso wi: N = S wi * Oi.
- Señal de salida: O = 0 si N £ Umbral
  O = 1 si N > Umbral.

A partir de esta "célula", se pueden construir redes neuronales, interconectando los axones y dendritas de

muchas células.

Para definir completamente una red neuronal, se tiene que especificar tanto el comportamiento individual de cada neurona como la interconexión existente entre ellas. Las neuronas que se comportan de forma similar se agrupan en capas.

Su cualidad principal es la capacidad para variar de comportamiento cuando cambia una situación.

```
R1: SI motor es V8
    ENTONCES el coche es un Ferrari 348.
R2: SI motor es V6 Y alerón es grande
    ENTONCES el coche es un Mitsubishi 3000.
R3: SI motor es V6 Y alerón es pequeño Y longitud es 4.40
    ENTONCES el coche es un Honda NSX.
R4: SI motor es V6 Y alerón es pequeño Y longitud es 4.52
    ENTONCES el coche es un Nissan 300.
```

El proceso mediante el cual aprende la red se denomina entrenamiento. El aprendizaje en una red consiste en ir cambiando los pesos de las conexiones adaptándola hasta conseguir la respuesta que se desea. Puede ser supervisado, en el que cada ejemplo estará formado por las entradas a la red y la salida que se espera para esas entradas, y no supervisado, en el que cada ejemplo de entrenamiento está constituido sólo por las entradas, sin dar la salida esperada, por lo que se asociará la información recibida con la almacenada (la memoria está formada por los pesos sinápticos entre neuronas).

```
R1: SI longitud es 4.23 ENTONCES coche es Ferrari 348.
R2: SI longitud es 4.40 ENTONCES coche es Honda NSX.
R3: SI longitud es 4.52 ENTONCES coche es Nissan 300.
R4: SI longitud es 4.56 ENTONCES coche es Mitsubishi 3000.
```

ALGORITMOS GENÉTICOS

Estos algoritmos buscan soluciones de forma adaptativa, en analogía a los cambios genéticos que se producen en las poblaciones naturales para adaptarse al medio. Comienzan con un conjunto de datos o patrones generados de forma aleatoria, y evolucionan hacia una solución a través de técnicas de adaptación de los patrones iniciales. La solución no será estrictamente la mejor, pero se puede estar muy cerca de ella.

Para representar las propiedades de un individuo o suceso se utiliza un patrón, en el que se indica la presencia (1) o ausencia (0) de cada propiedad. Hay un peso asociado a cada patrón (idoneidad), que indica su rendimiento en experiencias anteriores.

El problema a resolver se representa mediante una cadena compuesta por ceros y unos (101001101). Se genera aleatoriamente una población de individuos formados también por cadenas o patrones de este tipo. Se evalúa el rendimiento de cada patrón, aumentando su peso si clasifica de forma adecuada el caso o disminuyéndolo si no lo hace, y se seleccionan los individuos para la siguiente generación basándose en su rendimiento, reemplazando los que tengan menor peso. Entonces, se realizan cruces (intercambiando los bits) y mutaciones (cambiando un 0 por un 1, o viceversa) de forma aleatoria entre los patrones.

De esta forma, la población tenderá a múltiples copias de la cadena que tenga el rendimiento más óptimo.

UN POCO DE SOFTWARE

Para continuar con la construcción de un pequeño sistema experto, en esta ocasión junto al programa del artículo anterior, se entrega un sencillo motor de inferencia que permitirá razonar con la base de conocimientos que el lector confeccione (también se puede utilizar como base el fichero FRUTAS). El programa está realizado en Turbo C 2.0. El código fuente está en el fichero SISTEM1.C, y el ejecutable es SISTEM1.EXE. A simple vista, cuando se ejecuta el programa, lo único nuevo respecto a la versión anterior es la opción r: razonar con el conocimiento.

Eligiéndola, el "sistema experto" preguntará acerca de los atributos de un objeto contenido en la base de conocimientos. Los atributos son las condiciones de una regla, y el objeto que cumple esos atributos es la parte acción de la regla. Se puede responder con SI (s), con NO (n), o con POR QUÉ (p). En el último caso, el sistema va a explicar porqué pregunta por determinado atributo, especificando a qué objeto intenta llegar. También indica el contenido actual de la base de datos, distinguiendo entre los hechos afirmados y los negados.

La función razona() rastrea todas las reglas disponibles, enviando la parte acción y la lista de condiciones de cada regla a la función comprueba().

Se mantienen dos listas: síes, que contendrá las condiciones ya afirmadas y noes, que contendrá las condiciones ya negadas.

La función comprueba() verifica mediante la función es_condicion(), si se cumplen todas las condiciones (se deduce la parte acción) y mediante no_es_condicion(), si hay una condición que ya está negada (se descarta la regla en curso). Utiliza la función no_preguntada(), para pasar a otra condición de la lista si ya se ha preguntado por la condición en curso. Interroga al usuario acerca de determinada condición. Si responde afirmativamente, ésta se añade a la lista síes, y si responde negativamente, se añade a la lista noes. Cuando se cumplan todas las condiciones de una regla, la función comprueba() devuelve verdad a razona() y se infiere la parte acción, que será el objeto buscado. ■

BIBLIOGRAFÍA

* Frost, R.A.: Introduction to Knowledge Base Systems. Collins.

* Finder, Nicholas V.: Associative Networks. Academic Press.

* Hayes, Roth: Building Expert Systems. Addison Wesley.

* J.K., Deberham: Knowledge Systems Design. Advances in Computer Science Series. Pentice Hall (1989).

* Castillo, Álvarez: Sistemas Expertos. Métodos y Herramientas. Paraninfo.

* Louis E. Frenzel, Jr.: A fondo: Sistemas Expertos. Anaya.

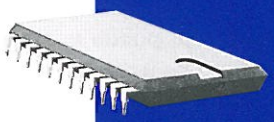
* Sierra, Guillermo J.:Sistemas Expertos en Contabilidad y Administración de Empresas. RAma.

* Ringland, Duce: Approaches to Knowlodge Representation. An Introduction. Research Studies Press Ltd.

* J.W. Grzymala-Busse: Manageing Uncertaing in Expert Systems. Kluwer Academic Publishers.

PROGRAMAS RESIDENTES BAJO DOS

José Galán



Seguramente todos los lectores asiduos de la revista han pensado, en alguna u otra ocasión, en la posibilidad de desarrollar un programa residente que realizase alguna función útil, o que incluso añadiera alguna capacidad nueva al sistema operativo DOS de Microsoft. Pero, una vez fijados los objetivos del proyecto, el lector se habrá encontrado con multitud de obstáculos impuestos, por un lado, por la falta de documentación acerca del desarrollo de este tipo de software y, por otro, por las limitaciones que impone al programador el diseño del DOS. A las circunstancias antes comentadas hay que añadir la falta de un estándar para la programación de TSR, lo cual hace que algunos de estos programas planteen problemas de incompatibilidad con otras aplicaciones de este tipo. Esperamos que con este artículo, y los que le sigan, los lectores puedan desarrollar, con un mínimo de problemas, cualquier tipo de programa residente para DOS.

INSTALACIÓN DE PROGRAMAS RESIDENTES

Como todos los lectores saben, la instalación de un TSR se suele limitar a la ejecución del fichero ejecutable, ya sea un EXE o un COM, tras lo cual el programa queda residente en el sistema.

Esta tarea que, desde el punto de vista del usuario, es trivial no lo es tanto desde el punto de vista del programador, ya que unas rutinas de instalación diseñadas con poco esmero pueden acarrear consecuencias como, por ejemplo, un consumo de memoria excesivo.

Se debe tener en cuenta que un TSR debe, después de haber instalado su parte residente, devolver el control al DOS de forma que el software que se ha instalado no sea destruido. Para realizar dicha tarea se debe devolver el control al DOS mediante la función 31h en lugar de mediante la habitual 4Ch que usan la gran mayoría de las aplicaciones.

El uso de la función 31h es sencillo, siendo necesario únicamente especificar en el registro DX la cantidad de párrafos de memoria que deben ser conservados.

Un programa residente debe finalizar mediante la función 31h y no con la habitual 4Ch

Personalmente, a la hora de desarrollar programas residentes, prefiero implementarlos en ficheros COM en lugar de crear un ejecutable EXE. La razón fundamental de dicha elección se debe a que los programas COM ocupan menos espacio que los EXE, tanto en disco como en memoria, ya que estos últimos deben disponer de la información necesaria para relocalizar los segmentos que los componen. Lo anteriormente comentado no quiere decir, en forma alguna, que no sea posible programar buenos residentes como EXE.

En el programa ejemplo LISTADO 1.ASM pueden observar los lectores un esqueleto de instalador de un TSR, op-

La programación de programas TSR ha sido siempre una tarea ardua debido a múltiples factores, como las limitaciones que impone el DOS y la dificultad que supone la depuración de este tipo de aplicaciones. Con la serie de artículos que se inicia en este ejemplar de Sólo Programadores, adquirirá los conocimientos básicos para programar sus propios TSR.

timizado para ocupar la menor cantidad posible de RAM una vez instalado.

Durante la instalación de la parte residente del programa se debe almacenar el contenido de todos los vectores de interrupción que vayan a ser usados en el programa ya que, como se verá más adelante, será necesario saltar al código al que apuntan en más de una ocasión.

Otra función vital que debe realizarse en esta parte es la comprobación de si el programa ya se encontraba instalado en el sistema. Una práctica común para realizar esta función consiste en chequear la presencia de una cadena ASCII determinada en una determinada posición de memoria. Para indicar dicha posición de memoria puede usarse el contenido de algún vector de interrupción que sea usado por el TSR. En algunos TSR la cadena ASCII que sirve de identificación se encuentra en el interior de una rutina de gestión de alguna interrupción hardware, como es el caso de algún programa residente de copia de disquetes en segundo plano.

Durante el proceso de instalación es muy conveniente comprobar la existencia de bloques de memoria superior disponibles o incluso de memoria expandida.

Una programación adecuada hace posible que el TSR pueda alojarse en dichas zonas de memoria, dejando más memoria convencional libre para la ejecución de aplicaciones.

ACTIVACIÓN DE UN TSR

Las formas de activar un programa residente son variadas, pero la que sin duda alguna plantea una mayor complejidad para el programador es la que hace uso de hot keys, o combinaciones de teclas, que mediante su pulsación deben producir la ejecución del servicio, o servicios, que el TSR proporciona.

El sistema más eficiente para detectar la pulsación de una serie de teclas requiere la gestión, por parte del programa residente, de la interrupción generada por el teclado cuando se pulsa o se suelta una tecla.

La gestión del hardware del teclado junto con el uso de una variable cuyos bits nos indican el estado de las teclas que forman nuestra hot key hace posible la activación de un programa residente en el momento en que el usuario lo necesite.

Sin embargo la captura de la interrupción del teclado no es un sistema infalible, ya que si dicha interrupción ha sido capturada por otro programa residente o por la aplicación que se ejecuta en primer plano nuestro TSR no tendrá oportunidad de leer los códigos de las teclas que sean pulsadas.

En el primer caso, con una programación adecuada, y cargando nuestro TSR posteriormente al causante de los problemas es muy posible que desaparezcan las incompatibilidades. Se hace necesario, por lo tanto, la siguiente práctica de buen comportamiento: si la tecla que ha sido pulsada, o soltada, no afecta a nuestro programa residente debemos pasar el control al gestor de interrupciones que se encontraba activo antes de instalar nuestro programa residente.

El archivo LISTADO2.ASM muestra un pequeño programa residente que sigue la norma de buen comportamiento anteriormente mencionada.

LIMITACIONES IMPUESTAS POR EL DOS

Lo primero que el programador debe tener en cuenta es que el MS-DOS es un sistema operativo monousuario y monotarea. Esto implica que todo el diseño del sistema se basa en que los servicios que el DOS proporciona a las aplicaciones están pensados para proporcionar su funcionalidad solamente a una aplicación. Debido a los anteriores preceptos el DOS es un sistema operativo que no es reentrante. Pero, ¿qué significa que el DOS no es reentrante?

Cuando un programa invoca alguno de los servicios del DOS, mediante una instrucción INT, las rutinas del sistema operativo utilizan una serie de variables en la memoria RAM del sistema. Ahora bien, supongamos que mientras se está ejecutando una rutina interna del DOS, que implementa la funcionalidad de un servicio del sistema operativo, un programa residente toma el control e invoca el mismo servicio del sistema que fue interrumpido por el TSR.

Lo que ocurrirá es que el TSR obtendrá del DOS el servicio que ha solicitado, pero a costa de machacar las variables internas del sistema que se estaban utilizando cuando el residente tomó el control. Cuando su TSR retorna al programa que fue interrumpido, en nuestro caso el DOS, las variables internas del sistema no contendrán valores correctos por lo que lo más probable es que se produzca un error del sistema de consecuencias impredecibles.

Durante la instalación es conveniente comprobar la existencia de bloques de memoria superior disponibles

Si el DOS fuese reentrante las rutinas de servicio del sistema crearían en la pila el espacio necesario para alojar las variables que necesitaran utilizar. De esta forma, si el sistema es llamado mientras se está ejecutando un servicio del mismo, las nuevas variables que se necesitan se crearían en la pila por lo que, al concluirse la ejecución del servicio invocado por el TSR, se restauraría el valor original del puntero de la pila (registro SP) y podría continuarse la ejecución del sistema operativo sin contratiempo alguno.

Todo lo anteriormente comentado hace necesario que los programas residentes tomen ciertas precauciones antes de llamar a rutinas del DOS, especialmente los servicios de ficheros que se invocan mediante INT 21h, para asegurarse de que no se está ejecutando una zona crítica del sistema operativo.

La forma más común, aunque indocumentada por Microsoft, para averiguar si es seguro llamar a una función del DOS se basa en la comprobación del valor de una variable interna del sistema operativo llamada InDosFlag. En el programa, en una de sus partes, se ilustra como se puede averiguar la dirección de dicha variable.

Para que no se produzcan efectos colaterales indeseados al llamar al DOS desde un TSR es preciso que el valor de InDosFlag sea cero. Sin embargo, si comprobamos continuamente su valor, nos daremos cuenta de que, normalmente, su valor es uno. Esto se debe a que el DOS, cuando espera a que el usuario introduzca algún comando, pone a uno el contenido de InDosFlag.

Entonces, ¿cómo detectar cuando el DOS está esperando a que el usuario introduzca un comando en la línea de comandos del sistema?. Para resolver está situación se dispone de una interrupción, INT 28h concretamente, que es llamada por el sistema cuando está esperando a que el usuario introduzca una orden en la línea de comandos.

Por lo tanto la forma más segura de detectar si es posible realizar una llamada al DOS es mediante una combinación de técnicas que consultan el estado de InDosFlag y que, al mismo tiempo, comprueban si el DOS está esperando un comando mediante la intercepción del vector de interrupción correspondiente a INT 28h.

PRECAUCIONES CON LA BIOS

Muchas veces los programas residentes deben averiguar, cuando son activados, determinada información sobre el estado del ordenador como, por ejemplo, el modo de vídeo que se encuentra activo.

Siempre que sea posible dicha información debe buscarse no realizando una llamada al BIOS mediante una instrucción INT, sino consultando la zona de variables del BIOS que se mantiene en memoria RAM y, por lo tanto, es accesible sin plantear problemas para todas las aplicaciones.

Otra posible fuente de problemas son los servicios de acceso a disquetes y disco duro de que dispone la BIOS y que, de forma continua, son usados por el DOS para acceder a dichos dispositivos.

Si en su TSR existe una zona de código cuya temporización es crítica y, por lo tanto, las interrupciones del procesador son inhabilitadas durante su ejecución debe comprobarse que el programa residente no ha interrumpido al código que presta los servicios de INT 13h.

Diversos tipos de software gestionan directamente las IRQs 0 y 1

¿Cómo comprobar que su programa residente no ha interrumpido a INT 13h?. Nuevamente la solución se basa en la captura de dicha interrupción. El gestor que debe implementar, en este caso, no es complicado ya que su única función es activar una variable que informará al programa residente de su activación mientras se está prestando un servicio de acceso a disco. En este caso, el comportamiento más seguro que puede adoptar el TSR es retornar al programa interrumpido y, quizás, ejecutarse posteriormente usando la técnica denominada ejecución diferida.

REENTRABILIDAD DEL PROPIO TSR

Supongamos que durante la ejecución del programa residente es pulsada, por segunda vez, la combinación de teclas encargada de activar el TSR. Si no se toman las debidas precauciones lo más probable es que, por segunda vez, se ejecute el residente.

Al igual que ocurre con el DOS lo más probable es que su programa residente tampoco sea reentrante, por lo que se debe evitar su ejecución cuando ya se esté ejecutando.

Este problema es de sencilla solución, bastando con que el programador defina un flag que tome un valor determinado cuando el programa residente se está ejecutando. Dicho flag deberá ser consultado por el código encargado de activar el TSR, de forma que resulte imposible la ejecución anidada del programa residente.

Al devolver el control desde el TSR basta con resetear el valor del flag, de forma que sea posible una posterior activación del programa residente.

EJECUCIÓN DIFERIDA

Como el lector se habrá percatado, después de la lectura de los párrafos anteriores, existen numerosas ocasiones en las que no resulta seguro la activación de un programa residente.

La ejecución diferida consiste en llamar al programa residente una vez que hayan finalizado los sucesos que hacían insegura su activación.

Para implementar esta funcionalidad se debe disponer de algún mecanismo que haga posible la llamada, a intervalos periódicos, a una rutina del programa residente que compruebe la posibilidad de ejecutar de forma segura el TSR.

Realizar esta tarea es sencillo, ya que el PC, como ocurre con prácticamente todos los ordenadores, dispone de un temporizador que genera interrupciones a intervalos regulares de tiempo.

Como todos los lectores de Sólo Programadores saben, el temporizador del PC, implementado mediante un chip 8253, dispone de tres contadores. De ellos el contador cero es el encargado de generar el reloj que es mantenido por el sistema operativo, mientras que el uno provoca el refresco de la memoria RAM mediante el subsistema DMA, y el dos se encuentra dedicado a la producción de sonidos mediante el altavoz que equipa a los PC.

La información sobre el sistema debe obtenerse consultando la zona de variables del B.I.O.S.

Cuando el DOS se está ejecutando el temporizador produce interrupciones a razón de, aproximadamente, 18 veces por segundo. Una opción consiste, por lo tanto, en la intercepción del vector de interrupción ocho, que es el correspondiente al temporizador.

Si el programador no desea realizar accesos al hardware en la rutina de gestión del temporizador, puede enganchar su gestor al vector 1Ch en lugar de al 8, ya que la rutina de la BIOS encargada de gestionar esta interrupción ejecuta una instrucción INT 1Ch antes de devolver el control.

Como siempre que se intercepta una interrupción que es usada por el sistema, o por otros programas residentes, es conveniente, en lugar de retornar con una instrucción IRET, hacerlo pasando el control al gestor de interrupciones que se hallaba instalado previamente, ya que de esta forma se da oportunidad a otros programas residentes para que se ejecuten.

LENGUAJES Y DEPURACIÓN DE PROGRAMAS RESIDENTES

Sin duda una de las etapas más peliagudas del desarrollo de un programa residente es su depuración. Esto es así ya que, en la mayoría de las ocasiones, no va a ser posible llamar a un debugger cuando está activado el programa residente mediante la pulsación de la combinación de teclas que forma el hot key.

Una técnica muy apropiada para el desarrollo de programas TSR, consiste en realizar la programación de la aplicación como si se tratase de un programa convencional no residente. Una vez que el proyecto ha sido totalmente depurado tan sólo debemos añadirle las rutinas necesarias para hacerle funcionar de forma residente.

Generalmente este último paso se limita a una serie de funciones que se encargan de comprobar si resulta segura la ejecución del programa residente y, si así ocurre, ejecutarle.

En cuanto a los lenguajes de programación más apropiados para el desarrollo de programas residentes el ensamblador es, sin duda alguna, el más apropiado ya que ofrece indudables ventajas como la velocidad de ejecución, y lo compacto del código producido que, en este caso, es una consideración vital ya que los programas residentes no disponen para ellos de toda la memoria del sistema.

Si no desea programar todo el proyecto usando ensamblador puede optar por desarrollar una librería en dicho lenguaje encargada de la instalación del programa y su ejecución de forma segura. De esta forma el resto del proyecto puede ser desarrollado en un lenguaje de alto nivel, como C o Pascal por ejemplo.

BIBLIOGRAFÍA

Es difícil encontrar documentación que explique, a fondo, el desarrollo de programas residentes, por lo que lo más conveniente es disponer de una serie de obras que cubran diversos aspectos de la programación del PC.

Las obras comentadas a continuación le han resultado al autor de este artículo de gran ayuda a la hora de programar varios programas residentes y de otro tipo y espero que también lo sean para el lector:

Solucionario del programador para IBM PC, XT, AT. Escrito por Robert Jourdain cubre gran parte de la programación del hardware de los compatibles, si bien al ser una obra algo antigua no entra en temas como, por ejemplo, la programación de la VGA.

MS DOS Avanzado. Escrito por Ray Duncan es todo un clásico que, en mi opinión, no debe faltar en la biblioteca de cualquier programador de compatibles PC.

PC Interno. Escrito por Michael Tischer es una magnífica obra a la que una penosa traducción la quita gran parte de su atractivo ya que, en algunos pasajes, este hecho unido a la total ignorancia del traductor del tema tratado hace que algunos pasajes sean totalmente ininteligibles. El libro dispone de un capítulo completo dedicado al desarrollo de programas residentes.

LIMITACIONES DE LOS PROGRAMAS RESIDENTES

Todos los métodos anteriormente explicados para la activación de un TSR, así como para su ejecución segura en el entorno DOS, no servirán de nada si deseamos activar el programa residente durante la ejecución de un programa que capture las interrupciones de teclado y del temporizador.

Diversos tipos de software, sobre todo videojuegos, gestionan directamente las IRQs cero y uno, correspondientes al temporizador y al teclado. Esta práctica que, por un lado, permite a los desarrolladores de juegos optimizar sus productos hace imposible la programación de software residente que pueda ejecutarse bajo cualquier tipo de circunstancia.

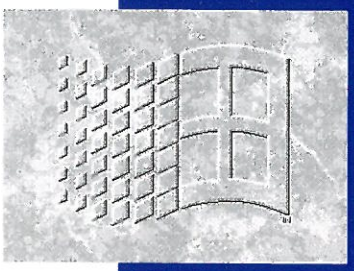
Otra posible fuente de problemas viene de la mano de las tarjetas gráficas. Actualmente muchos juegos utilizan los denominados modos X de la VGA que son, básicamente, modos gráficos no documentados por IBM, que hacen posible la obtención de mayores resoluciones con 256 colores o la utilización de varias páginas de vídeo. Esta situación, con un adecuado conocimiento de los registros del hardware VGA, puede ser solventada por el programador haciendo que, al activarse su TSR, se preserve el contenido de la pantalla de forma adecuada.

Más complicada de resolver es la situación que plantean los modos denominados SuperVGA. La disponibilidad de microprocesadores cada vez más potentes ha hecho que diversos desarrolladores de software pongan en el mercado productos que utilizan modos superVGA. La enorme cantidad de fabricantes y modelos de este tipo de hardware hace imposible al programador la preparación de una serie de rutinas preparadas para manejar todas las situaciones que se pueden llegar a plantear. Bajo este contexto la única esperanza que le queda al programador de software residente es que, en lugar de usar modos propios de cada hardware SuperVGA, se usen los modos definidos por VESA, los cuales se hallan bien documentados así como la forma de acceder al buffer de vídeo en dichos modos.

Como se ve el desarrollo de programas residentes no es una tarea sencilla, más bien todo lo contrario. El programador interesado en realizar este tipo de aplicaciones debe ser consciente de que como más se aprende es precisamente programando, y de que los problemas que encuentre en el desarrollo de sus primeros TSR le serán de un valor incalculable cuando afronte proyectos más ambiciosos. Espero, en todo caso, que los consejos que se han dado en este artículo sean de ayuda para todos aquellos lectores interesados en programar residentes, ya que las posibilidades que ofrecen este tipo de programas son francamente apasionantes. ■

CONTROLES DE EDICIÓN

Jorge R. Regidor



Dentro de los controles incluidos en Windows existe uno encargado de la edición de texto, es el control EDIT. Este control tiene una potencia singular, permitiéndonos realizar desde la entrada de texto en una sola línea, pasando por entrada de un password y llegando hasta un editor multilínea realmente potente. Además con una gran facilidad a la hora de programar. Las capacidades de EDIT vienen impuestas por los estilos indicados en su fase de creación.

Los controles EDIT se utilizan frecuentemente en la programación para Windows, tanto para realizar la entrada de unas palabras en un diálogo, como para realizar un auténtico editor de texto. En la Figura A , se ilustra los dos usos de dicho control.

ESTILOS DEL CONTROL EDIT

Como ya se ha indicado, la forma de definir las capacidades del control EDIT es mediante sus estilos. Con estos estilos creamos el control EDIT enfocándolo a nuestras necesidades particulares. Los estilos soportados se pueden dividir en cinco tipos según su funcionalidad y todos empiezan por las letras ES_:

I. Estilo de desplazamiento en pantalla.

ES_AUTOHSCROLL: El control tendrá una barra de desplazamiento horizontal para mover el texto.

ES_AUTOVSCROLL: El control tendrá una barra de desplazamiento vertical para mover el texto.

II. Estilo de alineación.

ES_CENTER: El texto estará centrado.

ES_LEFT: El texto estará alineado a la izquierda.

ES_RIGHT: El texto estará alineado a la derecha.

III. Estilo de formato de caracteres.

ES_LOWERCASE: Se realizará una conversión a minúsculas de todos los caracteres.

ES_UPPERCASE: Se realizará una conversión a mayúsculas de todos los caracteres.

ES_OEMCONVERT: Se realizará una conversión al set de caracteres activos en MS-DOS.

IV. Estilo para definir el tipo de editor.

ES_MULTILINE: Indica que el editor será multilínea. Si no se incluye este estilo, el editor será de una sola línea.

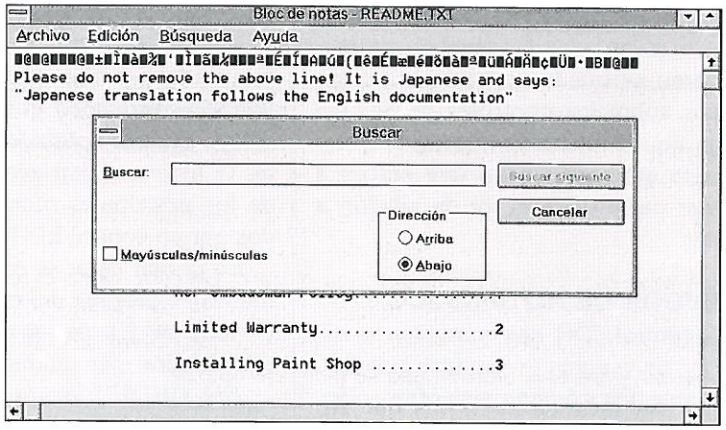


Figura A. Ejemplos de uso del control EDIT.

V. Otros estilos.

ES_NOHIDESEL: Indica que la selección de un texto no se pierda al perder el foco el control.

ES_PASSWORD: Indica que el texto introducido es secreto y por lo tanto no lo visualiza. Por defecto, se visualizan asteriscos.

ES_READONLY: Indica que el editor sólo permitirá leer el texto.

ES_WANTRETURN: Indica que editor debe introducir un retorno de carro al pulsar la tecla <Intro>.

Como se puede ver, existen diversos estilos con los cuales podremos hacer

Uno de los controles más potentes definidos en Windows es el control EDIT. En este control está casi integrado un sencillo editor de textos y este casi restante, se refiere a las capacidades para abrir y guardar ficheros, búsquedas de cadenas dentro del texto e impresión.

un editor a la carta. Además podemos añadir cualquier estilo general definido para las ventanas en general (WS_...).

La forma de combinar estos estilos dentro de la función CreateWindow es la normal, mediante el operador |. Valga el siguiente ejemplo:

```
hEditWnd = CreateWindow("EDIT",
NULL, WS_CHILD | WS_VISIBLE|
ES_MULTILINE |WS_VSCROLL
|WS_HSCROLL| ES_AUTOHSCROLL

|ES_AUTOVSCROLL,0,0,iWidthEdit,

iHeightEdit,hMainWnd,IDC_EDIT,
hInstance, NULL);
```

En el WorkShop de Borland C++ 4.0, para crear controles de edición existe una caja de diálogo para definir los estilos como la mostrada en la figura B.

Las capacidades del control EDIT están impuestas por sus estilos en el momento de su creación

Como se puede apreciar, todos los estilos sobre los controles de edición están aquí contenidos, siendo sólo necesario activarlos para que entren a formar parte del control de edición a definir.

CÓDIGOS DE NOTIFICACIÓN

El control EDIT nos comunica su estado y los eventos producidos sobre él, mediante los códigos de notificación. Estos códigos empiezan por las letras EM_ y se definen en la tabla A.

La forma de procesar estos códigos de notificación se realizará dentro del procedimiento de la ventana padre, ya sea está la ventana principal o una ventana de dialogo. El código es el siguiente:

```
switch(uMensaje)
{
 case WM_COMMAND:
 {
  switch(wParam)
  {
   case IDC_EDIT:
   {
    switch(HIWORD(lParam)
    {
     case EN_CHANGE:
      .
      .
     case EN_VSCROLL:
    }
   }
  }
 }
}
```

donde IDC_EDIT es el identificador del control, uMensaje es el código del mensaje recibido, y wParam y lParam son los parámetros WORD y LONG, respectivamente, del mensaje recibido.

MENSAJES PROCESADOS POR EL CONTROL EDIT

El control EDIT además de tener mensajes propios (aquellos que empiezan por EM_), gestiona por defecto algunos de los mensajes generales de Windows (aquellos que empiezan por WM_). En este apartado se va a describir la funcionalidad derivada del envío de los principales mensajes gestionados por un control EDIT.

En primer lugar se empezará por los mensajes propios del control. Existen un total de 33, de los cuales se van a describir los más utilizados.

EM_GETSEL devuelve los índices inicial y final de un texto seleccionado, tomando por índices la posición de cada carácter dentro del texto o buffer.

EM_SETSEL se encarga de marcar una posición entre dos índices.

EM_LINESCROLL desplaza el texto el número de filas y/o columnas indicado, EM_GETMODIFY indica si ha habido modificaciones en el texto.

EM_SETMODIFY pone el flag de texto modificado con el valor especificado.

EM_GETLINECOUNT devuelve el número de líneas del texto.

EM_SETHANDLE establece el buffer que contendrá el texto del control EDIT. Sólo con el estilo DS_LOCALEDIT.

EM_GETHANDLE obtiene el buffer donde se encuentra el texto del control EDIT. Sólo con el estilo DS_LOCALEDIT.

EM_LINELENGTH devuelve la longitud en bytes de la línea especificada.

EM_REPLACESEL reemplaza la selección actual con la cadena especificada.

EM_GETLINE transfiere una línea a un buffer suministrado.

EM_LIMITTEXT limita el número de caracteres de un control EDIT.

EM_CANUNDO indica si es posible gestionar el mensaje EM_UNDO con éxito.

EM_UNDO deshace el último cambio realizado.

EM_LINEFROMCHAR devuelve el número de línea que contiene un carácter dado.

EM_SETPASSWORDCHAR establece el carácter a visualizar en editores con estilo ES_PASSWORD. Por defecto es un asterisco.

EM_EMPTYUNDOBUFFER vacía el buffer de recuperación de cambios.

EM_GETRECT indica el tamaño y posición del control EDIT.

EM_SETRECT cambia el tamaño u posición del control EDIT.

EM_SETREADONLY pone la marca de texto de sólo lectura.

EM_SETTABSTOPS establece la posición de los tabuladores.

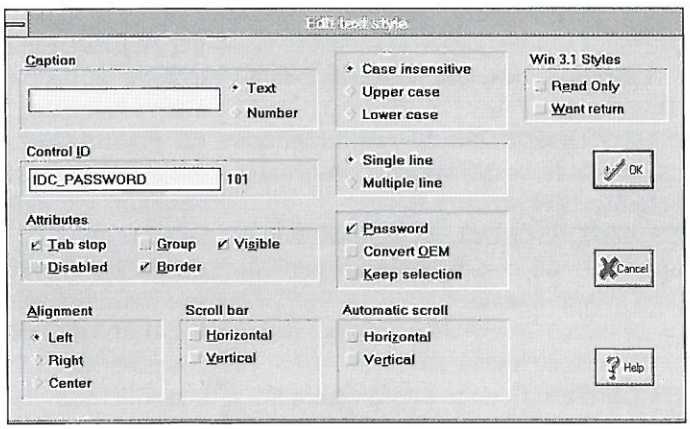


Figura B. Caja de diálogo con los estilos definibles del control EDIT desde WorkShop.

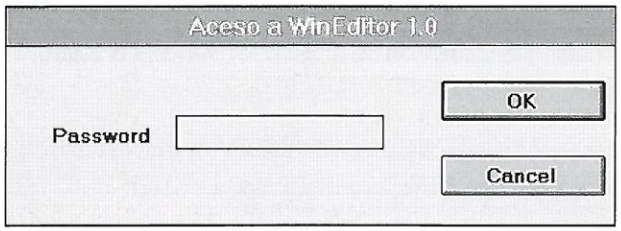


Figura C. Caja de diálogo con un control de edición de una línea para entrada de password.

Mediante todos estos mensajes se puede realizar muchas operaciones sobre el control EDIT, aunque además, se pueden realizar otras mediante los mensajes de ventanas como:

WM_SETFONT establece la fuente del editor.

WM_SETFONT devuelve la fuente utilizada por el editor.

WM_CUT corta y envía al portapapeles la selección realizada.

WM_PASTE inserta el contenido del portapapeles.

WM_COPY copia al portapapeles la selección hecha.

WM_CLEAR borra la selección hecha.

WM_CHAR escribe un carácter en el editor.

WM_GETTEXT copia el número especificado de caracteres a un buffer.

WM_GETTEXTLENGTH devuelve la longitud del texto.

WM_SETTEXT copia al editor el contenido del buffer especificado.

El formato de estos mensajes, así como de sus parámetros y valores de retorno se encuentran fácilmente en la ayuda del compilador.

Detallar cada uno de ellos sería añadir de forma redundante la información de la ayuda del SDK lo que implicaría varias páginas que a su vez se encuentran fácilmente localizables y de más fácil acceso. El objetivo fundamental es saber que existen, lo que hacen y cuando necesitemos utilizarlos, buscar su formato y profundizar en ello.

EL BUFFER DEL TEXTO.

Por defecto Windows cuando crea un control EDIT, también crea un buffer para el texto, el cual puede crecer como sea necesario hasta unos 32 Kb. Además se puede imponer otro limite mediante el mensaje EM_LIMITTEXT. Cuando alguno de los dos limites son sobrepasados, el control EDIT envía el mensaje de notificación EN_ERRSPACE.

Cuando se crea un control EDIT dentro de un diálogo, este buffer se crea fuera del segmento de datos de la aplicación. Si necesitamos que el buffer se cree dentro del segmento de datos de la aplicación se debe añadir el estilo DS_LOCALEDIT. Con este estilo, es trabajo de la aplicación actualizar el buffer de texto. Al inicio del diálogo, se debe mandar el mensaje EM_SETHANDLE, pasándose como parámetro el handle del buffer en memoria local creado con LocalAlloc. Si se produce una notificación de falta de espacio, debe ser atendida de la siguiente manera:

1. Se recibe el handle actual del buffer del control mandando el mensaje EM_GETHANDLE.
2. Se libera la memoria con la función LocalFree.
3. Se obtiene un nuevo buffer con la función LocalAlloc.
4. Se manda el nuevo handle del buffer al editor con el mensaje EM_SETHANDLE.

Las aplicaciones que no hagan uso del estilo DS_LOCALEDIT, no deben llamar a los mensajes EM_SETHANDLE o EM_GETHANDLE.

PROGRAMA DE EJEMPLO

Para crear un programa de ejemplo sobre el control EDIT la elección no era otra que crear un sencillo editor de texto. Para ello y junto con las dos siguientes entregas del curso, en los que se añadirá capacidades de acceso a ficheros y control de menús y aceleradores, se creará el programa WinEditor. En una primera fase y dentro de esta entrega, se crea el núcleo del editor para ilustrar el uso del control EDIT. Este uso se enfoca en sus dos variantes como son dentro de una caja de diálogos y como ventana hija.

Todas las referencias en este apartado hacen referencia al código fuente (WINEDIT.C), por lo cual no sería mala idea tenerlo presente en el momento de la lectura.

Para el ejemplo del control EDIT como parte de una caja de diálogo se ha creado un diálogo para introducir un password de acceso al editor. Esto implica que dentro de la definición de estilos del control se ha activado la opción Password. La forma de crear el control desde Workshop es realmente sencilla y dentro de la programación , el acceso a las propiedades del control se hacen mediante las funciones GetDlgItem, GetDlgItemText, SetDlgItemText, SendDlgItemMessage... Cabe destacar dentro del código para gestionar el control EDIT, la forma que existe de indicarle el número máximo de caracteres que puede tener el password. Este se realiza enviando el mensaje EM_LIMITTEXT al control IDC_PASSWORD, que es el identificador del control EDIT dentro del diálogo. Por otra parte, es importante resaltar la forma de obtener el texto del control, ya que se hace me-

La comunicación con el control se realiza enviando mensajes y atendiendo a los códigos de notificación

Como cualquier otra ventana, el control EDIT gestiona algunos de los mensajes de control de ventanas de Windows

diante la función GetDlgItemText la cual normalmente sirve para obtener el titulo de un control y que en el caso del control EDIT, se utiliza para obtener el texto de dicho control.

Para el control EDIT hijo de la ventana principal el proceso de creación se hace desde la función CreateWindow, pasándose como nombre de la clase de ventana la cadena EDIT, se le pasan NULL como título de la ventana, los estilos necesarios para crear un control hijo, multilínea, con barras de desplazamiento y con memoria dentro del heap del programa, las dimensiones en principio pueden ser cualquiera aunque las más indicadas son el tamaño de área de cliente de la ventana padre, se le pasa el handle de la ventana padre, el identificador del control y la instancia. Si el control no se crea correctamente, se debe salir del programa, no sin antes eliminar la ventana padre para no ocupar memoria.

Para establecer la fuente del editor, se manda un mensaje WM_SETFONT a la ventana del editor (hEditWnd), con un handle a la fuente en el parámetro wParam y TRUE si se quiere redibujar el control.

Otro punto a destacar está ya dentro de la función de proceso de mensajes de la ventana padre, cuando ésta reciba el mensaje WM_SETFOCUS de activación del foco (la entrada del teclado se direcciona hacia esta ventana), se debe a su vez redireccionarla mediante la función SetFocus hacia el editor. De no hacerse así, al iniciarse el programa, se deberá pinchar con el ratón sobre el editor para comenzar a escribir. El otro mensaje que se tiene que atender dentro de la ventana principal es WM_SIZE, que nos indica que el tamaño de la ventana ha cambiado. Aquí se debe llamar a la función MoveWindow, para actualizar el tamaño del editor y que esté en concordancia con la ventana padre. El lector puede eliminar esta llamada para ver el efecto.

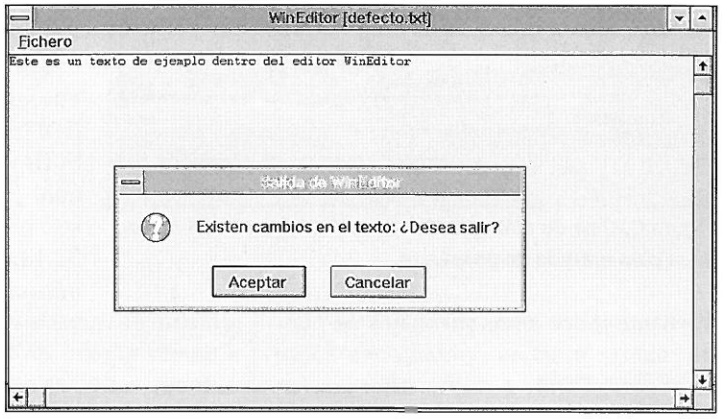


Figura D. Ventana principal con el control de edición en su área de cliente.

En el código se ilustra además la forma de atender los mensajes de notificación del control dentro de la función de tratamiento de mensajes de la ventana padre.

Como se puede observar, los códigos de notificación están incluidos como la parte alta del parámetro lParam del mensaje WM_COMMAND, además el parámetro wParam contiene el identificador del control.

El modo de mandar mensajes al control es como a cualquier otra ventana, mediante la función SendMessage. En el código se ilustra esto, mandando el mensaje EM_GETMODIFY para obtener el estado del texto (si ha sido o no modificado).

PROCESO DE COMPILACIÓN

Junto con el articulo se suministra la información para crear un proyecto en Borland C++ 4.02. Sin embargo, si utiliza otro entorno las modificaciones necesarias sobre el código fuente serán mínimas o ninguna, debido a ser el código compatible con el SDK para Windows 3.1.

CONCLUSIÓN

En este artículo se ha pretendido hacer un acercamiento a la programación de los controles de edición. Se ha pretendido dar una idea general de como crear y gestionar estos controles tanto desde una caja de diálogos como desde una ventana normal, así como los mensajes y códigos de notificación principales dentro de este control.

Por último, hacer un avance del siguiente articulo, dónde se hablará en profundidad de las cajas de diálogos y se creará una ampliación de este editor de textos, añadiendo capacidades de acceso a ficheros y búsqueda de palabras para ilustrar la programación de las cajas de diálogos en sus dos formas modales y amodales. ■

CÓDIGO	INFORMACIÓN
EN_CHANGE	Existe un cambio en el contenido del control.
EN_ERRSPACE	No existe suficiente memoria dinámica para la operación.
EN_HSCROLL	Se ha pulsado la barra de desplazamiento horizontal.
EN_KILLFOCUS	Se ha perdido el control de las entradas del teclado.
EN_MAXTEXT	Se ha sobrepasado el máximo de caracteres impuesto.
EN_SETFOCUS	Se ha recuperado el control de las entradas del teclado.
EN_UPDATE	Se precisa redibujar el control.
EN_VSCROLL	Se ha pulsado la barra de desplazamiento vertical.

Tabla A. Códigos de notificación del control EDIT.

CORREO LECTORES

Para cualquier duda referente a los temas/artículos tratatos en la revista, escriba una carta a:

Sólo Programadores
Referencia: *Correo Lectores*
TOWER COMMUNICATIONS
C/ Marqués de Portugalete, 10 Bajo
Madrid 28027

COMO SUSCRIBIRSE A

SÓLO PROGRAMADORES

Revista práctica para usuarios de PC

Suscribase enviando este cupon por correo o fax (91 320.60.72), o llamando al teléfono (91) 741.26.62 Horario 9 a 14 y 15:30 a 18:30 h.

Deseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año + regalo (filtro monitor) **por sólo** 10.995 ptas. ☐ Estudiantes carreras técnicas 40% Dto. (sobre precio portada): 8.250 ptas.

Nombre y apellidos.....Domicilio.....

Población.....C.P .....Provincia.....Telf.....Profesión.....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº .....

Fecha de caducidad de la tarjeta.....Nombre del titular, si es distinto.....

☐ Domiciliación bancaria.

Señor Director del banco .....

Población .....

CODIGO CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta de ahorro.

el recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L. como pago de mi suscripción a la revista SÓLO PROGRAMADORES.

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

Firma:

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Marques de Portugalete 10, Bajo
28027 Madrid.

MODOS SUPERVGA

David Aparicio

En un PC con MS-DOS, disponemos de la clásica pantalla de texto, del propio entorno Windows, y del modo gráfico a pantalla completa tan común en todos los juegos. En Linux es frecuente utilizar los equivalentes a los dos primeros, mediante las consolas virtuales y con X-Windows, pero el tercero es poco conocido. En este artículo exploraremos algunos aspectos del mismo.

Los juegos, principal forma de contemplar gráficos a pantalla completa, son por ahora poco frecuentes en Linux. Es por esto que la explotación de nuestra SuperVGA se ha limitado al uso del sistema X de ventanas. Sin embargo, el sistema ofrece una potente librería para construir aplicaciones gráficas con el que podremos sacar partido, de forma profesional, a este aspecto de nuestro PC.

Las nuevas reglas que se necesitan para programar en una máquina multiproceso no disminuyen las posibilidades gráficas de nuestra aplicación, sino que únicamente obligan a administrar racionalmente los recursos disponibles. Como prueba de ello, DOOM, el conocido arcade de acción, dispone de una versión para consola Linux tan rápida como su pariente DOS, quedando todavía potencia de sobra para poder ejecutar otros programas simultáneamente.

LOS COMPONENTES

Todos los ficheros necesarios para instalar la librería se encuentran en el disco 5 de la serie D de la distribución del CD del mes de febrero. El archivo se denomina svgalib.tgz. Bien mediante el programa setup o manualmente, se instalarán los siguientes componentes:

- la librería (en /lib y /usr/lib)
- las cabeceras de funciones (vga*.h)
- documentación (en /usr/doc/svgalib).

... ejemplos (en /usr/lib/svgalib).

PROGRAMACIÓN COOPERATIVA.

La característica multitarea de Unix introduce ciertas peculiaridades a la hora de manejar la pantalla. Dado que, mediante las consolas virtuales, podemos alternar entre varias pantallas, un programa no puede acceder indiscriminadamente a la VGA..

En primer lugar, el acceso al hardware está oculto por la librería. Esto permite estandarizar la programación, y nos ahorra además el esfuerzo de manejar las características propias de cada tarjeta.

En segundo lugar, los programas deben ser diseñados teniendo en cuenta que no están sólos en el sistema. Antes de dibujar en pantalla deben comprobar, por ejemplo, si disponen del control de la pantalla o si se encuentran en segundo plano. ¿Para qué gastar potencia en dibujar unos gráficos que no se están viendo?.

Por otra parte es preferible orientar los programas a eventos. En el sistema operativo DOS, los programas suelen permanecer en un bucle, leyendo continuamente en algún puerto y esperando alguna actividad. Esto produce el efecto de que, tanto cuando el programa está realizando algo importante, como cuando está simplemente esperando datos, está gastando capacidad de CPU que podría ser empleada con más provecho por algún otro programa.

En un sistema orientado a eventos, mientras no se presiona el teclado o se mueve el ratón, el programa permanece "dormido". En el momento que el usuario produce una entrada, el programa recupera el control, procesa la acción, y vuelve a quedar inactivo.

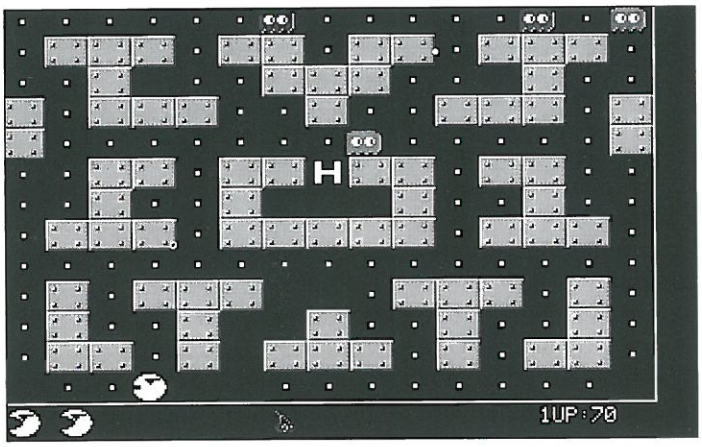
Una vez vista la filosofía que se requiere para realizar una programación eficiente, pasaremos a enumerar los diferentes componentes de la librería que nos ocupa.

FUNCIONES DE MANEJO DEL RATÓN

El siguiente grupo de funciones permiten actuar sobre el ratón. Pasamos a enumerar las mas frecuentes:

mouse_init:

Indicamos el dispositivo asociado (normalmente /dev/mouse), el tipo, y el numero de veces que se explora su estado. Mediante esta función activamos el ratón. El tipo del ratón se elige



LUXMAN, una versión del conocido PACMAN.

explícitamente mediante alguna constante de la librería, o bien con la función de autodetección vga_getmousetype.

mouse_close:

Para indicar que finaliza el manejo del dispositivo. Normalmente se invoca al terminar un programa.

mouse_waitforupdate:

Al llamar a este procedimiento, el sistema no devuelve control hasta que no se ha movido el ratón, o se ha pulsado una tecla.

mouse_update:

Igual que la anterior, pero la función retorna inmediatamente, y devuelve un valor de tipo boolean indicando si hay algún cambio disponible o no.

mouse_getx, mouse_gety, mouse_getbutton:

Obtiene las coordenadas del ratón, y el estado de los botones. Se utiliza MOUSE_LEFTBUTTON, MOUSE_MIDDLEBUTTON y MOUSE_RIGHTBUTTON para discernir cada botón. Por ejemplo, para saber si el botón derecho esta pulsado:

```
t = mouse_getbutton();
if(t&MOUSE_RIGHTBUTTON) {
  /* botón derecho pulsado */
  }
else {
  /* botón derecho sin pulsar */
  }
```

mouse_setposition:

Para posicionar el ratón en un punto concreto.

mouse_setxrange, mouse_setyrange:

Para indicar los valores entre los que puede oscilar el movimiento del ratón, tanto en el eje horizontal como en el vertical.

mouse_seteventhandler:

Instala una función de usuario para que registre los movimientos del ratón.

mouse_setdefaulteventhandler:

Reinstala el manejador de la librería para controlar el ratón.

Los parámetros exactos y las constantes asociadas a las llamadas se pueden observar en el fichero 'vgamouse.h'.

FUNCIONES DEL CONTROL DE TECLADO.

Para leer de un teclado, disponemos de uno de los siguientes métodos:

- Modo normal (cooked): El sistema lee líneas completas, interpretando caracteres de edición (cursores y secuencias de control). Se devuelven los caracteres asociados a las teclas presionadas. Este método es el más usual, y se emplea al invocar la tradicional getch() en C.
- Modo crudo (raw): El sistema devuelve las pulsaciones del teclado tanto al presionar como al soltar las teclas. El sistema devuelve los códigos de las teclas (scancode), o bien los caracteres asociados a ellas. El primero es un código que depende únicamente de la posición de la tecla en el teclado. Por ejemplo, podemos producir el código 39 al presionar cierta tecla. Un mapa de teclado español asociará a dicho número el carácter 'ñ', mientras que otro americano devolverá el carácter ';'. Esta relación depende simplemente de una conversión software, denominada 'remapeo del teclado'.

Teniendo en cuenta todo esto, podemos indicar aquí que, dependiendo de la llamada de librería que se emplee, obtendremos códigos de rastreo (llamadas de nivel más bajo) o caracteres (llamadas de nivel superior). Las enumeradas a continuación son llamadas de manejo en forma 'scancode':

keyboard_init:

Inicialización del teclado en modo 'código de rastreo'.

keyboard_close:

Retorno del teclado a modo normal.

keyboard_update:

Comprueba la situación actual del teclado.

keyboard_keypressed:

Indica si la tecla asociada a cierto código esta pulsada o no.

La siguiente llamada de la librería es independiente de las anteriores, y retorna caracteres al estilo tradicional:

vga_getch:

Retorna inmediatamente, con -1 o con la última pulsación. Si se ejecuta en combinación con las llamada keyboard_init, retorna scancodes. En otro caso devuelve los caracteres asociados a las teclas pulsadas.

El fichero donde encontraremos las funciones y constantes asociadas se denomina 'vgakeyboard.h'.

FUNCIONES GRÁFICAS BÁSICAS

Se agrupan aquí las llamadas relacionadas con el manejo del propio hardware, y con los diferentes modos gráficos asociados. Podemos distinguir algunas llamadas, como:

vga_init:

Inicializa la librería, y establece valores de partida. Se debe invocar al iniciar el programa.

vga_setmode:

Establece el modo gráfico que indiquemos. Por ejemplo, vga_setmode(G640x480x256) pone la pantalla en modo 640x480, con 256 colores. Por otro lado, vga_setmode(TEXT) establece el mo-

do normal de la consola, y se debe utilizar al acabar el programa. Podemos consultar todos los modos disponibles en la propia documentación.

vga_getxdim, vga_getydim, vga_getcolors:

Interroga sobre la capacidad de puntos y colores del modo gráfico actual.

vga_getmousetype, vga_getmonitortype, vga_hasmode, vga_getcurrentchipset:

Funciones de autodetección del hardware disponible en nuestro sistema.

vga_lockvc, vga_unlockvc:

Bloqueo y desbloqueo de la consola cuando se están efectuando operaciones críticas con la VGA, impidiendo que el usuario cambie de pantalla.

vga_runinbackground:

Indica al sistema que nuestra aplicación distingue si está en la pantalla activa o no, mediante la siguiente llamada. Si no se invoca, el programa es detenido cuando se cambia de consola, y se reanuda su ejecución al volver a la original. De este modo, el sistema impide que el proceso acceda a la VGA mientras esté en segundo plano.

vga_oktowrite:

Interroga al sistema sobre si la consola asociada a nuestro programa está activa o no. Mediante esta llamada y la anterior, el programa coopera con el resto del sistema multitarea, con objeto de aprovechar al máximo los recursos disponibles. Un ejemplo de uso de las últimas llamadas es el siguiente:

```
vga_runinbackground(1);
/* inicialización */
while(1) {
 if(vga_oktowrite()) {
   vga_lockvc();
   /* operaciones gráficas */
   vga_unlockvc();
   }
 /* otras operaciones del bucle */
 }
```

vga_waitretrace:

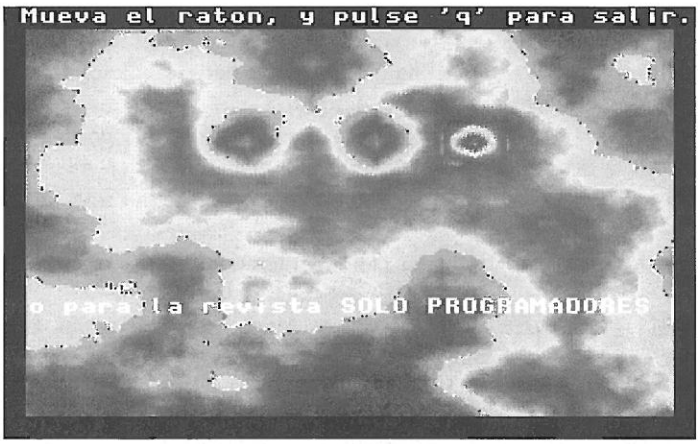
Permite disminuir parpadeos en el trazado, sincronizando nuestro programa con la señal de refresco del monitor.

graph_mem:

A diferencia del resto, esto no es una función, sino un puntero a la memoria de vídeo, para poder escribir directamente en pantalla. Es aprovechada ampliamente por las funciones avanzadas, y por cualquier librería construida a partir de las llamadas de la librería svgalib.

vga_getch:

Espera la lectura de un carácter, y devuelve lo leído. A diferencia de la llamada estándar getch, no espera a leer la línea completa (vea 'modo normal' en la sección de teclado). No se requiere el uso de las funciones del grupo 'keyboard_'.



Una pantalla del programa de ejemplo.

vga_getkey:

Devuelve un carácter, o -1 si no hay tecla pulsada. Es útil para explorar el teclado de vez en cuando, mientras se realizan otras operaciones en la pantalla.

La documentación que incluye la librería explica estas funciones, y también se puede revisar el fichero vga.h para comprobar los parámetros adecuados.

FUNCIONES GRÁFICAS AVANZADAS

En el apartado anterior hemos echado un vistazo a las llamadas básicas de la librería para SuperVGA. Podemos intuir, con razón, que reúne un conjunto de primitivas demasiado simplificadas para lo que nosotros deseamos: dibujar. Por ello, se han implementado funciones construidas recubriendo parte de las llamadas anteriores, y que permiten utilizar las capacidades gráficas de nuestro sistema sin tener que realizar ninguna operación complicada. Veamos a continuación algunas:

gl_setcontextvga:

Inicializa la librería de funciones avanzadas. Se debe invocar antes que ninguna otra de este grupo.

gl_setpixel, gl_getpixel:

Permite leer o escribir un punto de la pantalla al color deseado.

gl_line, gl_circle:

Dibuja líneas y circunferencias, del color que se indique.

gl_fillbox:

Rellena un área rectangular con el color que se indique.

gl_getbox, gl_putbox:

Lee y escribe áreas rectangulares de pantalla, para permitir animaciones.

gl_copybox:

Copia una zona rectangular de una posición de pantalla a otra.

gl_setclippingwindow, gl_enableclipping, gl_disableclipping:

Permite limitar el área visible de nuestros dibujos a una cierta ventana de pantalla.

gl_setfont, gl_write:

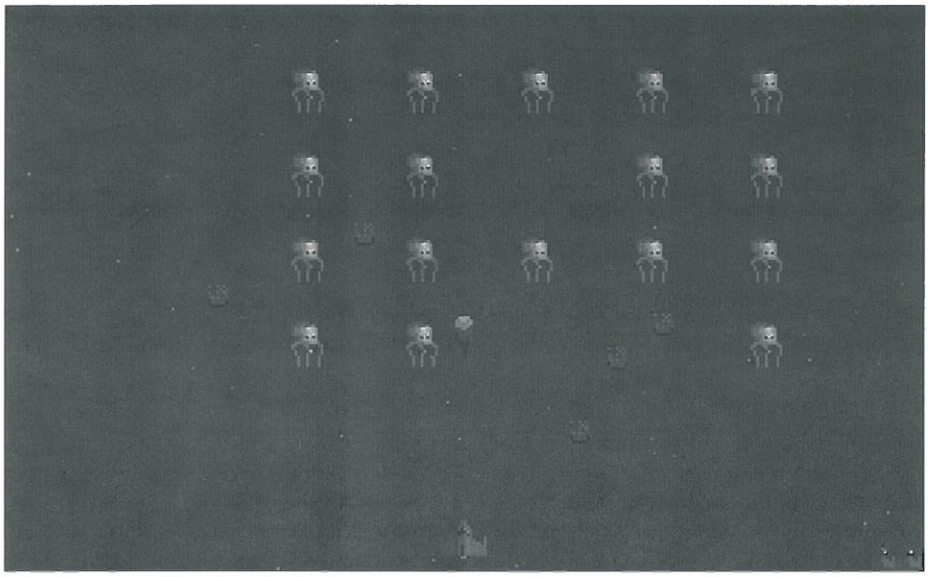
Permite seleccionar y escribir con diferentes tipos de letra, en modo gráfico.

Mediante estas sencillas llamadas, y otras más complicadas que se encuentran descritas en la documentación y enumeradas en vgagl.h, podemos realizar vistosas aplicaciones gráficas.

COMPILACIÓN DE PROGRAMAS

Para construir un programa que utilice un modo gráfico deberemos incluir la cabecera donde se encuentren las llamadas que se empleen:

```
/* Funciones generales, familia "vga_" */
#include <vga.h>
/* Funciones de dibujado, familia "gl_" */
#include <vgagl.h>
/* Manejo de teclado. Funciones "keyboard_" */
#include <vgakeyboard.h>
```



ZAP'EM, un matamarcianos.

```
/* Manejo de ratón. Funciones "mouse_" */
#include <vgamouse.h>
```

Repase los ejemplos incluidos en el disco, para ver la estructura empleada.

Al compilar, simplemente añada la opción "-lvga" para indicar al compilador que utilice la librería gráfica (sólo llamadas "keyboard_", "mouse_" y "vga_").

```
cc -o miejec miprog.c -lvga
```

Si se utilizan llamadas "gl_", se debe invocar la opción "-lvgagl -lvga".

```
cc -o miejec miprog.c -lvgagl -lvga
```

PROBLEMAS DE CONFIGURACIÓN

Al compilar nuestro primer programa, quizá uno de los ejemplos que se incluyen en el disco de la revista, nos podremos encontrar con algunos problemas leves. Básicamente podemos distinguir dos de ellos:

Mensaje de error "svgalib:Cannot get I/O permissions"

Los programas que utilizan los modos avanzados de nuestra tarjeta gráfica deben acceder a recursos hardware delicados, principalmente los registros de programación y bancos de memoria de nuestra tarjeta. Debido a esto, se requiere que el superusuario "dé el visto bueno" a los programas compilados. En palabras técnicas, se debe ejecutar con identificador de usuario root. Para cada ejecutable, deberemos habilitar los permisos adecuados. Supongamos que tenemos un programa 'miprog'. Como superusuario, ejecutamos:

```
chown root miprog
chmod u+s miprog
```

A partir de ahora el programa ejecutará con máximo nivel de privilegio, y el error debería desaparecer.

Si no ocurre así, es posible que tengamos una versión de kernel de Linux inferior a 1.1.59. En dicho caso, es necesario actualizarlo.

Mensaje de error "svgalib: Segment violation"

En general, este error indica un acceso a memoria fuera del rango asignado al proceso. En otras palabras, se está escribiendo aleatoriamente en la memoria, el modo protegido 386 la ha detectado, y el programa ha sido cancelado. Básicamente, esto indica que se están utilizando datos sin inicializar.

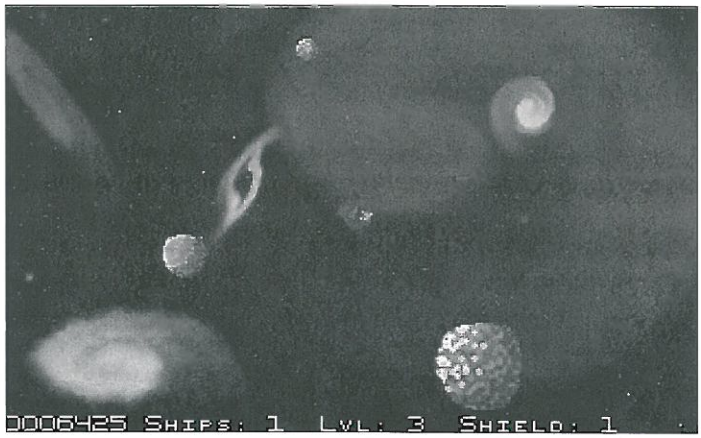
En el caso que nos ocupa, probablemente hemos olvidado llamar:

- vga_init
- vga_setmode
- gl_setcontextvga

Las dos primeras funciones inicializan datos para las funciones de la librería vga. La última inicializa datos para la librería vgagl.

CONCLUSIONES

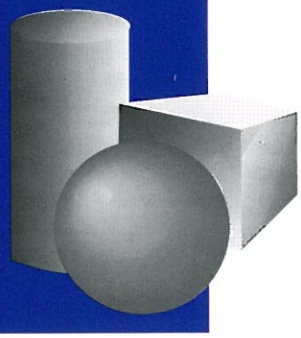
Se ha dado aquí una pequeña introducción al manejo de la librería de alta resolución. Sin embargo, la potencia (y complejidad) de las funciones que incluye van más allá del objetivo de este artículo. Si desea profundizar en el tema, se recomienda la lectura de bibliografía especializada. Una vez que conceptos como "registros extendidos", "bancos de memoria", "triple buffer"... le resulten familiares, descubrirá que dispone de funciones que permiten controlar de forma profesional el comportamiento de su aplicación. (sincronización con el refresco de pantalla, escritura a buffer de previsualizado y mucho más).



SASTEROIDS, al estilo del viejo clásico.

Si por el contrario no le suenan los conceptos anteriores, y le ha asaltado el comprensible sentimiento de no desear conocerlos nunca, se proporciona funcionalidad equivalente a las clásicas librerías gráficas que acompañan a los compiladores de casas tan populares como Borland y Microsoft. Podrá dibujar vistosas presentaciones sin tener que acudir a una programación complicada.

En definitiva, y como siempre, experimente y disfrute. ■



SISTEMAS DE VENTANAS EN C++

Ignacio Cea

Los algoritmos y técnicas empleadas en la construcción de un sistema de ventanas como Windows es uno de los secretos mejor guardados que hay dentro de la informática. En este serie de artículos vamos a tratar de desvelar los más relevantes, usando como lenguaje de programación el C++. No pierda detalle.

Este artículo, como primero de una serie de ellos, trata de mostrarle la base para el desarrollo de un sistema de ventanas; es decir, los algoritmos y técnicas que permiten a las ventanas moverse por entre un grupo de ellas sin que eso parezca afectar mucho al rendimiento general del sistema. En esta edición se verá, concretamente, las técnicas de reconstrucción de un sistema de ventanas cuando alguna de ellas varía su posición o tamaño.

Las ventanas de los ejemplos son vistas como simples objetos con forma de rectángulo susceptibles de visualizar información en su interior. No se tienen, por tanto, en cuenta los bordes, cabeceras, barras de iconos, etc... que completan, por ejemplo, la típica ventana Windows. Tampoco se tiene en cuenta el hecho de que una ventana pueda desplazarse por el interior de otra (ser hija de otra) como ocurre, por ejemplo, con cualquiera de los "grupos de programas" que integran la ventana del "administrador de programas" en Windows.

LAS VENTANAS... EN ORDEN

Entre en cualquier sistema de ventanas (por ejemplo Windows) y observe. Las ventanas, a pesar de estar situadas dentro de un entorno en dos dimensiones (su pantalla) dan la sensación de estar colocadas unas delante de otras (tres dimensiones).

El sistema de ventanas se comporta, la mayor parte del tiempo, como si de una "pila" se tratara; es decir, la última ventana que se crea es la que aparece en la parte superior de todas; mientras que la creada en primer lugar está en la parte más baja.

Bien es cierto que este funcionamiento es habitualmente alterado, por ejemplo, cuando colocamos una de las ventanas situadas por debajo de la principal en la cima de todas presionando el botón del ratón en una zona visible de la misma. Ésta, y otras formas de comportamiento semejante, no pueden asemejarse al de ninguna estructura dinámica y se van a considerar, simplemente, como excepciones al funcionamiento genérico de "pila" inicialmente mencionado.

Puede que esté pensando en cómo encaja este modelo cuando tenemos dos ventanas completamente visibles situadas una al lado de la otra. Para alguien que vea esta situación sin haber seguido todo su proceso de creación y colocación, es imposible determinar quien de las dos está por encima de la otra. Sin embargo estará por encima aquella que fue abierta en última instancia. De hecho estará remarcada de una manera diferente (por ejemplo, un color distinto en la cabecera).

UNA PILA DE VENTANAS

Queda claro pues, en principio, que necesitamos una estructura que mantenga el orden en el que las ventanas están situadas dentro de la pantalla y que ésta puede ser representada mediante una "pila". Pero, ¿qué información se ha de guardar en cada uno de los nodos de la misma?: Además de los datos necesarios para el manejo de la estructura (puntero al siguiente elemento), es lógico pensar que podría considerarse una referencia al objeto ventana situado dentro de esa posición o a un conjunto de enteros que contuvieran las coordenadas de las esquinas del mismo o a una mezcla de ambos.

```
Clase NodoVentana
class NodoVentana {
        private :
                NodoVentana *Siguiente;
                NodoVentana *Anterior;
                Ventana *Propietario;

        public :
                NodoVentana (Ventana
*prp,NodoVentana *ant = NULL,
                NodoVentana *sgt = NULL);
                ~NodoVentana ();
                inline NodoVentana *si-
guiente (void) {
                return (Siguiente); }
                inline NodoVentana *ante-
rior (void) {
                return (Anterior); }
                inline Ventana *propietario
(void) {
                return (Propietario); }
};
```

Figura 1.

Para contener esa información se ha creado la clase NodoVentana cuyo fichero .hh queda reflejado en pseudocódigo dentro de la figura 1.

REPINTANDO EL SISTEMA

Para reconstruir un sistema de ventanas ante cualquier cambio que se produzca dentro del mismo (movimiento de la ventana principal, por ejemplo) existen varias posibilidades.

La más simple de todas consiste, sin duda, en repintar, en su nueva ubicación y una por una, todas las ventanas del sistema recorriendo la "pila" de almacenamiento en sentido inverso; es decir, comenzando por la que éste en la parte más baja de la misma.

Ni qué decir tiene que este algoritmo es inviable dentro de sistema serio y mucho menos si, además, está pensado para funcionamiento en modos gráficos de alta resolución donde los recursos de pintado son demasiado costosos en tiempo. Si bien es cierto que suele ser el más empleado en los ejemplos didácticos de libros y revistas (Turbo C++ LADD), aunque éstos emplean el modo texto para hacerlo (figura 2).

```
// Recorre las ventanas dibujándolas
for (NodoVentana *nodo = Ventana::Inferior;
        nodo != Ventana::Superior; nodo = no-
do -> siguiente ())
  (nodo -> propietario ()) -> DIBUJAR ();
```

Figura 2. Reconstrucción del sistema de ventanas.

HABLANDO EN SERIO

No cabe duda de que lo más serio sería repintar sólo aquellas partes que se han alterado ante el cambio. Suponga un sistema, por ejemplo, con dos ventanas en el que una de ellas cubre la mitad de la otra (figura 3). Si el usuario mueve la ventana número 1, 100 pixels hacia la derecha, bastará con repintar sólo los 100 pixels de la ventana 2 que han quedado de nuevo visibles para tener todo el sistema reconstruido.

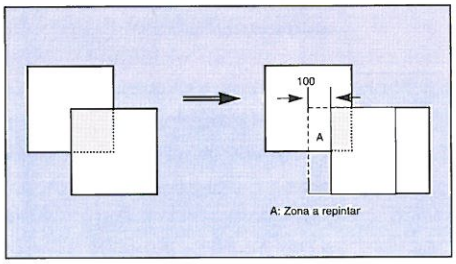


Figura 3. Moviendo una ventana.

Para poder hacer esto, cada ventana debería tener cierta información sobre las distintas zonas de otras ventanas que está tapando. De esta forma si aquella, por ejemplo, se mueve podrá recalcular con relativa facilidad qué partes de cada ventana tapada son ahora visibles y así repintarlas. Tras esto la ventana habría de recalcular qué partes son, de nuevo, ocultas y volver a empezar.

Es decir, que la solución parece estar en pasar la mayor parte del tiempo realizando cálculos sobre intersecciones entre rectángulos. Sin embargo, es preferible esto a tener que repintar mil y una veces toda una ventana en la que hay infinidad de líneas, círculos o bitimages.

LAS ZONAS TAPADAS POR UNA VENTANA

El conjunto de zonas tapadas por una ventana se van a almacenar dentro de un objeto de la clase ZonasEscritura situado en cada una de las distintas ventanas del sistema, objetos, a su vez de la clase Ventana.

```
Las otras clases
class NodoZona {
        private :
        NodoZona *Siguiente;
        NodoZona *Anterior;
        int X1,Y1,X2,Y2;
        Ventana *Propietario;

 public :
        NodoZona (int x1,int y1,int x2,int
y2,Ventana *prp,
        NodoZona *ant = NULL,NodoZona *sgt
= NULL);
        ~NodoZona ();
        inline NodoZona *siguiente (void) {
        return (Siguiente); }
        inline int x1 (void) { return (X1); }
        inline int y1 (void) { return (Y1); }
        inline int x2 (void) { return (X2); }
        inline int y2 (void) { return (Y2); }
        inline Ventana *propietario (void) {
                return (Propietario); }
};

class ZonasEscritura {
        private :
                NodoZona *Primero;
                NodoZona *Ultimo;

        public :
                ZonasEscritura (void) {
Primero = Ultimo = NULL; }
                ~ZonasEscritura ();
                void limpiar_objeto (void);
                void añadir_zona (int x1,int
y1,int x2,int y2,
                Ventana *Prp);
        inline NodoZona *primero (void) {
                return (Primero); }
        inline NodoZona *ultimo (void) {
                return (Ultimo); }
        void zonas_vistas_por (int x1,int y1,int
x2,int y2);
        void zonas_tapadas_por (int x1,int y1,
                                int x2,int y2);
};

class Ventana {
        private :
        static NodoVentana *Inferior;
        static NodoVentana *Superior;
        static void zonas_ocultas (void);

        ZonasEscritura ZonasTapadas;
                int X1,Y1,X2,Y2;

        public :
        Ventana (int x1,int y1,int x2,int y2);
                ~Ventana ();
                void mover (int ax,int ay);
                void reescalar (int ax,int ay);
                virtual void DIBUJAR () = 0;
                ...
};
```

Figura 4.

Cada una de las zonas tapadas, en cuestión, serán objetos de la clase NodoZona, en la que se almacenará información relativa al tamaño de la misma y la ventana a la que corresponde, así como datos que permitan

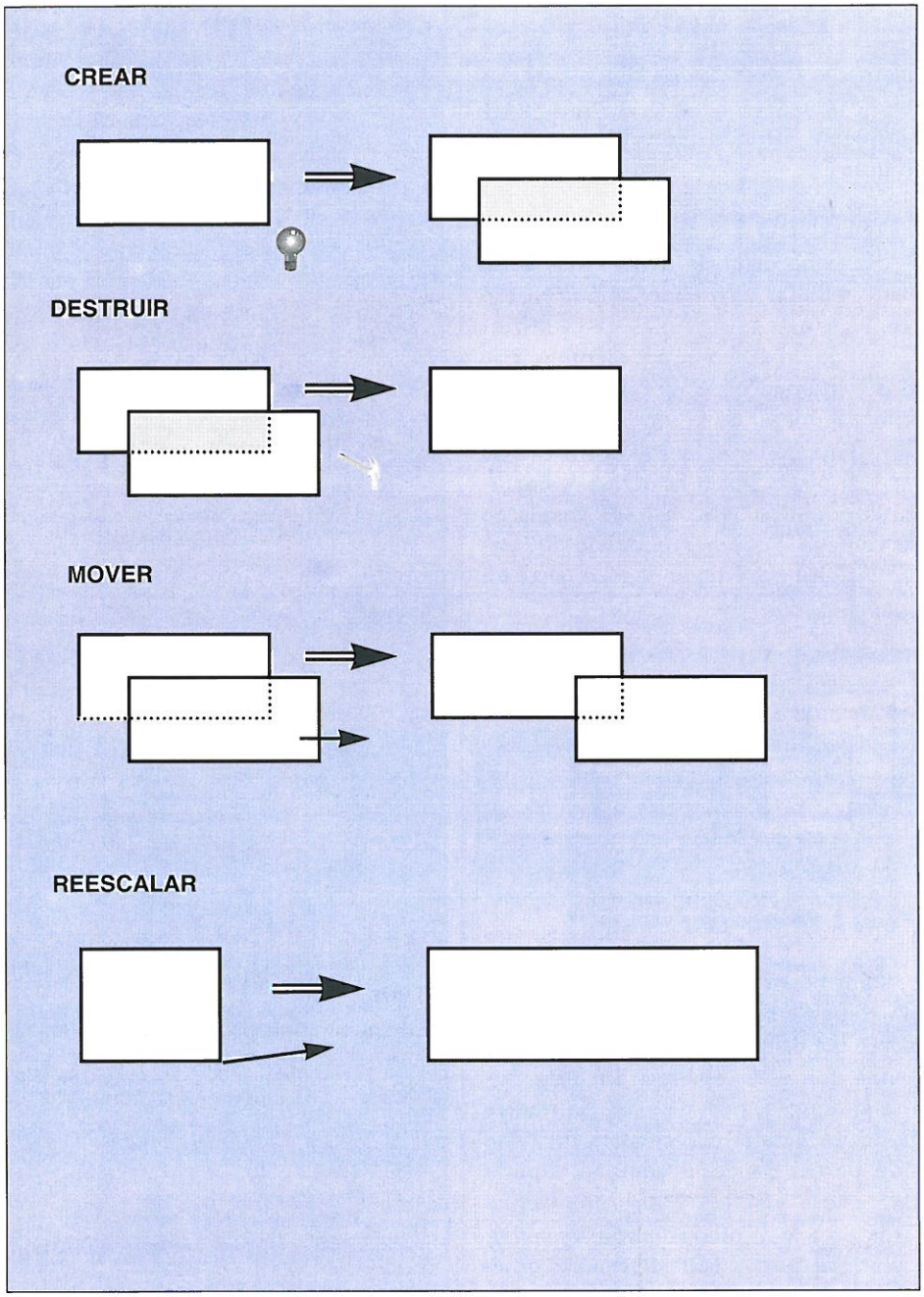


Figura 5. Acciones sobre las ventanas.

el enlazado de varios de ellos dentro de una estructura dinámica de información.

Dentro de la figura 4, se muestran los ficheros cabecera de cada una de ellas y las relaciones que unen a todas.

Siguiendo con el ejemplo inicial: Tanto la ventana 1 como la 2 albergan un objeto de la clase ZonasEscritura. El de la segunda estará vacío ya que no existe ninguna ventana por debajo suya, mientras que el de la primera, por el contrario, contendrá, en este caso, un sólo objeto de la clase NodoZona correspondiente a la parte sombreada que se muestra en la figura 1.

JUGANDO CON LAS VENTANAS

De entre todas las acciones que pueden llegar a hacerse con una ventana caben destacar las siguientes: crearla, destruirla, moverla o reescalarla (figura 5).

La primera de ellas, creación, supone calcular las distintas zonas (objetos NodoZona) de las otras estructuras ya existentes que son ahora tapadas por ésta. Zonas que formarán parte del objeto ZonasEscritura de la nueva ventana. La segunda de ellas, destrucción, implica recorrer todas esas zonas tapadas inicialmente calculadas y pintarlas.

La tercera y cuarta de ellas, por el contrario, son más complicadas ya que cuando una ventana se mueve o reescala suceden dos cosas: Por un lado que se dejan al descubierto partes de las zonas antes ocultas que deben ser repintadas para reconstruir por completo el sistema y, por otro lado que se tapan otras nuevas.

Por lo tanto, queda claro que necesitamos dos clases de algoritmos para sostener todas estas acciones. El primero de ellos, se empleará en el cálculo de las zonas tapadas por una ventana y el otro para calcular y dibujar las partes de las anteriores que quedan vistas ante un movimiento de la propietaria. Tanto uno como el otro parecen afectar sólo a la ventana objeto de la acción.

UN ALGORITMO DE OCULTACIÓN

El cuerpo del primero de los dos algoritmos, empleado durante la acción de creación, consiste inicialmente (ya se mejorará) en un sólo bucle que recorrerá todas y cada una de las ventanas ya existentes (sin contar con la que va a ser creada), desde la situada en la parte inferior hasta la superior. En cada paso se ha de calcular la zona de intersección entre la ventana apuntada por el bucle y la que va a ser creada y añadirla al objeto ZonasEscritura que hay dentro de la nueva ventana, ya que estos rectángulos son, a fin de cuentas, las zonas ocultas de esas otras ventanas. En la figura 6 se muestra un pseudocódigo para este algoritmo.

```
BUCLE DESDE ventana = Ventana::Inferior
        HASTA ventana = anterior a this
                Añadir dentro de
this.ZonasTapadas el resultado de la intersección
entre ventana y this;
FIN DEL BUCLE
```

Figura 6. Algoritmo inicial de ocultación

ALGORITMO PARA CALCULAR ZONAS VISTAS

El segundo de los algoritmos, empleado al cambiar la posición o tamaño de una ventana, será un simple bucle

extendido a todas las zonas de pantalla tapadas por la ventana objeto de la acción (calculadas en el momento de la creación de la misma). Para cada una de ellas se calcula la parte vista ante una intersección con un rectángulo cuyas coordenadas límite respondan a las nuevas características de la ventana propietaria. Esa parte vista calculada pertenecerá, sin duda, a otra ventana a la que será necesario invocar para que dibuje el trozo antes

```
BUCLE DESDE zona = ZonasTapadas.Primera ()
          HASTA zona = ZonasTapadas.Ultima ()
                    Hallar los rectángulos vistos
en la intersección entre zona y this (ventana);
                    Dibujar el resultado del an-
terior cálculo;
FIN DEL BUCLE
```

Figura 7: Algoritmo para el repintado de zonas vistas.

hallado. El pseudocódigo se muestra en la figura 7. Es necesario destacar que la parte vista de la intersección entre dos rectángulos no es, la mayor parte de las veces, otro; sino una forma compleja formada por la unión de varios (figura 8).

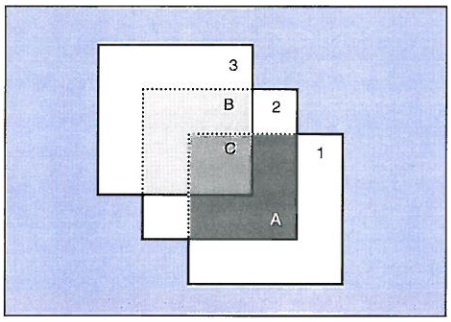


Figura 9: 3 ventanas.

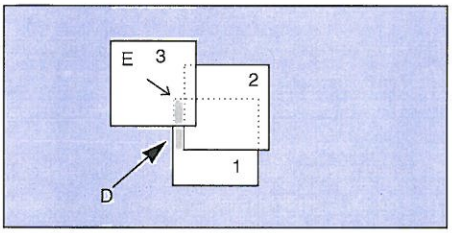


Figura 9. Bis.

Observe la figura 9. Según el algoritmo, las zonas ocultas de la ventana 1 no existen, la ventana 2 sólo tiene una, marcada como A, mientras que las de la 3 son la B (con respecto a la ventana 1) y la C (con respecto a la ventana 2).

Si la ventana 2 se moviera, ahora, hacia, por ejemplo, la derecha. Habría que reconstruir el trozo D, que según el

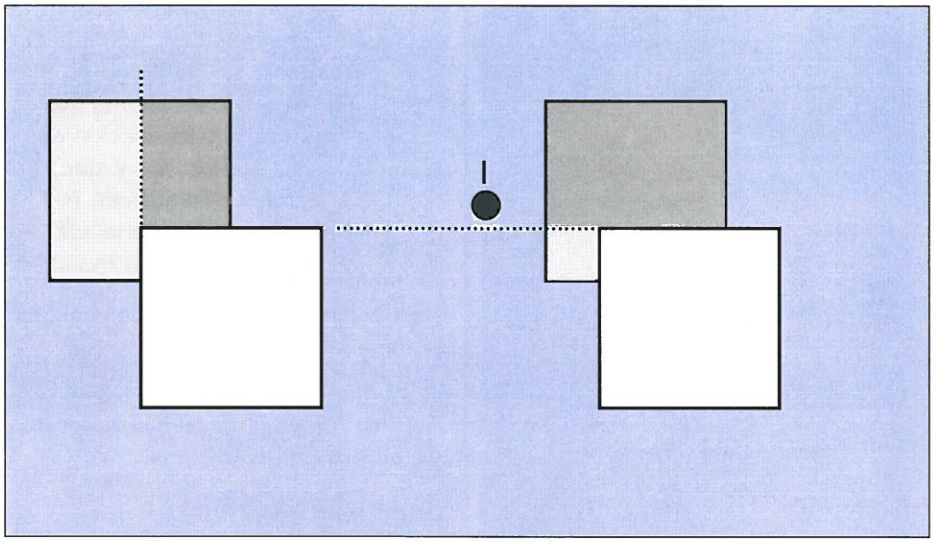


Figura 8. Partes vistas en la intersección de dos ventanas.

UN ALGORITMO IMPERFECTO

Los algoritmos hasta ahora descritos funcionan muy bien sobre un sistema de dos ventanas como máximo, pero el primero de ellos presenta una gran fisura si el número de estructuras manipuladas fuera superior. Veámoslo con un ejemplo.

algoritmo sería por completo de la ventana 1 tapada; sin embargo, hacerlo supone pintar una parte de la ventana 1 que está, a su vez, tapado por la ventana 3 (E), por lo que el resultado del movimiento sería desastroso. En general, realizar cualquier acción de movimiento o reescalado con una ventana

```
Algoritmo A: Con fisuras
// Elimina las zonas antigüas
ZonasTapadas.limpiar_objeto ();
for (NodoVentana *nodo = Ventana::Inferior;
          (nodo -> Propietario ()) != this;
          nodo = nodo -> siguiente ())

  // Añade la ventana completa a las zonas...
  ZonasTapadas.anadir_zona (nodo -> x1 (),nodo ->
y1 (),
          nodo -> x2 (),nodo -> y2 (),nodo -> pro-
pietario ())

// Calcula las partes tapadas...
ZonasTapadas.zonas_tapadas_por (X1,Y1,X2,Y2);

Algoritmo B: El óptimo
void Ventana::zonas_ocultas (void)
{
 ////////////
 // ACCION1 //
 ////////////
 // Bucle para recorrer todas las ventanas...
 NodoVentana *nodo_actual = Ventana::Inferior;
 do {

Ventana *ventana_actual = nodo -> propietario ();
          ZonasEscritura zonas_visibles;
          NodoVentana *nodo1 = Ventana::Inferior;
          do {

 Ventana *ventana1 = nodo1 -> propietario ();
           zonas_visibles.añadir_zona (ventana1 ->
X1,
                    ventana1 -> Y1,ventana1 ->
X2,ventana1 -> Y2,ventana);
          NodoVentana *nodo_patron = nodo1;
          while (nodo_patron != nodo_actual) {

          Ventana *ventana_patron =
                    nodo_patron -> propietario ();
                    zonas_visibles.zonas_vis-
tas_por (ventana_patron -> X1,
                    ventana_patron -> Y1,
                    ventana_patron -> X2,
                    ventana_patron -> Y2);
          nodo_patron = nodo_patron -> siguiente ();

           }

 // Añado las zonas vistas a la lista de ocultas...
          int x1,y1,x2,y2;
               while    (zonas_visibles.extraer
(x1,y1,x2,y2) != NULL)
                    ventana_actual                ->
ZonasTapadas.añadir_zona (x1,y1,c2,y2);

           // Prosigo con las vistas de la siguiente
ventana...
          nodo1 = nodo1 -> siguiente ();

          }

////////////
// ACCION 2 //
////////////
// Las zonas vistas han de ser partidas por la actual...
          ventana_actual                          ->
ZonasTapadas.zonas_tapadas_por (
                    ventana_actual -> X1,
                    ventana_actual -> Y1,
                    ventana_actual -> X2,
                    ventana_actual -> Y2);

          ////////////
          // ACCION 3 //
          ////////////
          // Las zonas realmente ocultas no las ta-
pan los de arriba
          for (NodoVentana *nodo_limite = nodo_ac-
tual -> siguiente (); nodo_limite != NULL;
nodo_limite = nodo_limite -> siguiente ()) {
Ventana *ventana_limite = nodo_limite -> propietario
();
ventana_actual -> ZonasTapadas.zonas_vistas_por (
                    ventana_limite -> X1,
                    ventana_limite -> Y1,
                    ventana_limite -> X2,
                    ventana_limite -> Y2);
          }

// Hace lo mismo con las siguiente ventanas...
          nodo_actual = nodo_actual -> siguiente ();

 } while (nodo_actual != NULL);
}
```

Figura 10. Zonas tapadas.

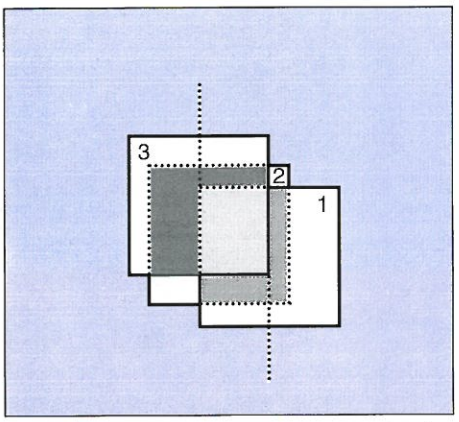


Figura 11. 3 ventanas (Sus zonas ocultas).

que no sea la superior supondría un desbarajuste completo en la integridad del sistema de ventanas.

UN ALGORITMO ÓPTIMO

La solución consiste en complicar el algoritmo de cálculo de zonas tapadas por una ventana. El algoritmo, ahora, es básicamente un bucle extendido a todas las ventanas del sistema. Con cada una de ellas se llevan a cabo tres acciones: En primer lugar, calcular las zonas vistas de aquellas ventanas que se encuentran por debajo de la señalada por el bucle. En segundo lugar, escoger las partes tapadas por la ventana actual sólo de entre las anteriores. Y, por último, en tercer lugar, elegir sólo aquellas partes de las zonas tapadas que no vayan a ser, a su vez, ocultadas por alguna de las que están justo por encima de ella en el sistema de ventanas. Este proceso hay que hacerlo cualquiera que sea la ventana que cambia. El código de este algoritmo se muestra en la figura 10.

Observe la figura del ejemplo 11. Suponga que queremos calcular las zonas ocultas de la ventana 2. Como dice el algoritmo, primero hemos de saber cuáles son todas las zonas que quedan vistas por debajo de esa ventana. En este caso es, simplemente, toda la ventana 1. En segundo lugar, hay que seleccionar las zonas ocultas de entre ellas; esto es, la zona A. Y por último, se han de recorrer todas esas zonas tapadas (sólo A) deduciendo sólo las partes que no van a ser ocultadas por la ventana 3. El resultado, por tanto, son los rectángulos A y B. El mismo principio puede ser aplicado para deducir las zonas tapadas por la ventana 3 (C, D y E).

EL PROCESO COMPLETO

Cuando la ventana 2 se mueva hacia abajo y la derecha, por ejemplo 10 pixels, no tenemos más que, tal y como indica el segundo algoritmo, calcular qué trozos de esas anteriores cajas (figura 11, rectángulos B y C) quedan vistas al ser superpuestas por la nueva ventana y pintarlas. Sin duda que el tiempo de dibujado que necesita este nuevo algoritmo es infinitamente más bajo, que el que requería el inicial.

El primitivo era, en realidad, algo propio de la ventana (dependía de ella). Sin embargo, el nuevo hay que extenderlo a todas las ventanas sea cual fuere la afectada por el cambio.

DISTRIBUCIÓN DE RESPONSABILIDADES

La responsabilidad de calcular las zonas tapadas por una ventana debe residir dentro de la misma ventana pero, ya que es un proceso que abarca todas las ventanas, sea cual sea la afectada por la acción, se va a implementar dentro de un método estático dentro de la clase Ventana.

Dentro de la clase ZonasEscritura se van a colocar dos métodos: el primero de ellos se ocupará de calcular qué zonas de los rectángulos almacenados quedan a la vista cuando son tapados por otro cuyas coordenadas se pasan como parámetro. Este método será empleado por las acciones 1 y 3 del algoritmo para el cálculo de las zonas tapadas por una ventana. El otro método, hará lo mismo que el anterior pero con las zonas ocultas y es empleado por la acción 1. Vuelva a mirar, ahora, todas las figuras relativas a los algoritmos.

```
Construir Ventana::Ventana (int x1,int y1,int x2,int
y2)
{
        ...
        Ventana::zonas_ocultas ();
}

Destruir
Ventana::~Ventana ()
{
        ...
        Algoritmo Figura 7;
        ...
}

Mover
void Ventana::mover (int ax,int ay)
{
        ...
        Algoritmo Figura 7;
        X1 += ax; Y1 += ay; X2 += ax; Y2 +=
ay;
        Ventana::zonas_ocultas ();
        ...
}

Estirar
void Ventana::estirar (int ax,int ay)
{
        ...
        Si la ventana se estira
          return;

        Algoritmo Figura 7;
        X2 += ax; Y2 += ay;
        Ventana::zonas_ocultas ();
        ...
}
```

Figura 12. Accionando Ventanas.

ACCIONES SOBRE VENTANAS

Ya es posible establecer las formas en las que las distintas acciones que tienen lugar sobre una ventana se llevan a cabo. Los códigos se muestran en la figura 12.

MEJORAS

Muchas de las cosas reflejadas en este artículo podrían haber tenido un enfoque mucho más dedicado a la filosofía de objetos; sin embargo el autor estimó que, teniendo en cuenta que estos ejemplos son la continuación de un reciente curso sobre C++ y que, por tanto, los lectores no tendrían, por regla general, excesiva experiencia, era mejor introducir el complejo mundo de los entornos de ventanas de una forma más suave. En cualquier caso se invita a todos a introducir todas aquellas mejoras que les suscite su imaginación. Aún así rogamos disculpas a nuestros lectores más puristas.

PRÓXIMO CAPITULO

En la próxima entrega nos ocuparemos de los algoritmos empleados para poder dibujar sobre ventanas; sobre todo en el caso de que la elegida no esté en la parte superior del sistema y se encuentre, además, parcialmente tapada por otras.

El código del sistema de ventanas que hemos empezado aquí a desarrollar será ampliado con esos otros nuevos algoritmos. ■

CAPTURA DE LA SALIDA DE UN COMANDO TSO A FICHERO

José María Peco

Tradicionalmente el lenguaje de CLIST es utilizado por el personal de sistemas para preparar, entre otras cosas, las utilidades y el entorno que necesita el personal que desarrolla aplicaciones. Pero muchas veces el desarrollador tiene ciertas necesidades que, por no conocer este lenguaje, no puede realizar por si mismo.

El presente artículo tiene una doble finalidad, poner a disposición del desarrollador una utilidad, y enseñarle el manejo de las instrucciones básicas de un procedimiento de comandos o CLIST.

La CLIST que se presenta a continuación tiene por objetivo capturar la salida generada por la ejecución del comando de TSO : LISTALL o LISTA, el cual presenta una lista de todas aquellas librerías a las que tiene acceso el que ejecuta el comando. Este comando es para el MVS lo que el PATH es para el MS-DOS.

Para su comentario se ha preferido fragmentarla a fin de tener a la vista las sentencias en su entorno.

```
/* <<<<<<<<<<                          JMPALO00                    00000100
 >>>>>>>>>>>>>>>>>>>>>> */
/* ========== CAPTURA Las librerías alocadas  ========*/              00000200
   PROC 0                                                            00000300
   CONTROL MSG                                                       00000400
   SET CABECERA = <<< Librerías alocadas >>>                         00000500
   SET RC = 0                                                        00000600
   CONTROL NOLIST NOCONLIST                                          00000800
   SET &SYSOUTTRAP = 999                                             00000900
   LISTA                                                             00001000
   SET &SYSOUTTRAP = 0                                               00001100
   SET &NOMBRE = TABLA                                               00001200
   SET &CONTA  = 0                                                   00001300
   SET &ULTIMO = &SYSOUTLINE                                         00001400
```

En esta primera parte de la CLIST se ejecuta el comando y se inicializan variables.

La línea :

0900 limita a 999 el número de líneas a capturar.

1000 Ejecuta el comando de TSO: LISTA.
Este comando es el que genera una salida por pantalla con la lista de las librerías que tiene alocadas el usuario que ejecuta el comando.
Esta salida no se puede redireccionar a un fichero, y es éste, por lo tanto, el objetivo de esta clist:
CAPTURAR LA SALIDA GENERADA SOBRE PANTALLA POR UN COMANDO DE TSO QUE NO SE PUEDE REDIRECCIONAR A UN FICHERO.

1010 Reinicializa el número de líneas a capturar.

1020 Iniciativa la variable NOMBRE con el valor TABLA

Tradicionalmente en los grandes sistemas existen dos mundos raramente integrados, el mundo del personal de sistemas, y el mundo del personal encargado del desarrollo de las aplicaciones. El presente artículo va dirigido a este último, que trabaja en el entorno del TSO, y que debido al divorcio que existe entre ambos mundos no dispone de las utilidades que, seguramente, el personal de sistemas tiene para realizar este cometido.

```
/*                                                              00001500
                    */
 /*                 ========= MEMORIZAR SALIDA ========== */   00001600
                    /*
*/                                                              00001700
*/
SET &CONTA          = &CONTA                                    00001800
 LOOP1:
                    IF  &CONTA > &ULTIMO THEN GOTO LISTAR       00001900
                    SET   &NOMBRE&CONTA =                       00002000
                    SYSOUTLINE&CONTA
                    GOTO LOOP1                                  00002100
```

1030 Inicializa la variable CONTA con el valor 0

1040 Memoriza el valor contenido en la variable &SYSOUTLINE, el cual especifica el número de líneas reales capturadas.

En esta segunda parte, se carga una variable indexada con cada una de las líneas memorizadas. Esto es necesario ya que el próximo comando que se ejecute inicializa las variables del sistema SYSOUTLINEnnn

La línea:

1800 Incrementa el contador de líneas CONTA.

1900 Examina si el número de línea es mayor que el número de líneas capturadas, el cual se encuentra contenido en la variable ULTIMO, y en el caso de que sea así, bifurca al párrafo etiquetado como LISTAR.

2000 Carga en la variable de usuario TABLAnnn el contenido de la variable del sistema SYSOUTLINEnnn que contiene la línea nnn capturada en la ejecución del comando.

2010 cede el control al párrafo etiquetado como LOOP1.

El siguiente grupo de instrucciones tiene por objetivo la definición y alocación del fichero que contendrá la salida.

la línea:

3100 El nombre del fichero de salida se compone del alias del usuario que ejecuta la clist (contenido en la variable del sistema &SYSUID) seguido del cualificador .SALIDA

3800 Esta sentencia examina si existe un fichero catalogado que tenga por DSN el nombre contenido en la variable FICHERO.
En el caso de que exista, la función &SYSDSN devuelve el valor OK, y por lo tanto solo hay que:

3900 Alocar el Dataset, asignarle el nombre lógico de SALIDA para el tratamiento que viene
Definir su disposición: OLD (existe y se borran los datos actuales)
Definir si se puede volver a usar en esta sesión : REUSE

4000 En el caso de que no exista el fichero con el nombre especificado, este comando se encarga de alocar un fichero en el sistema con el nombre dado, con la disposición NEW CATALOG y como es nuevo, hay que especificar:
- el tipo de unidad sobre la que alocar el fichero,
- formato y longitud del registro
- Tamaño del bloque y espacio en disco
Así mismo, también se define como reusable este fichero para esta sesión.

4400 Abre para salida el fichero que tiene por nombre lógico SALIDA

4500 Mueve el Texto : "LA SALIDA CONTIENE:" al buffer asociado al fichero de salida: SALIDA

4600 Escribe en disco el contenido del buffer SALIDA.

4800 Inicializa la variable RC (Código de retorno) de usuario con 0

La siguiente secuencia se corresponde a la rutina de tratamiento de errores que será aplicable desde el momento en que se produzca alguna situación de excepción.

La rutina de errores comienza por la palabra clave ERROR y engloba todas las instrucciones contenidas entre DO y END (líneas 5000 a 6100)

```
/*                                                              00003000
                    */
LISTAR:             SET &FICHERO= &SYSUID..SALIDA               00003100
R:
*/                                                              00003500
                    */
/*                  ====== ESCRIBIR SALIDA EN FICHERO          00003600
                     ====*/
/*                                                              00003700
                    */
                    IF &SYSDSN('&FICHERO') = OK THEN +          00003800
                                                                00003900
                    ALLOC FILE(SALIDA) DATASET('&FICHERO') OLD
                    REUSE
                    ELSE ALLOC FILE(SALIDA) DATASET('&FICHERO') 00004000
                    NEW CATALOG +
                                                                00004100
                    UNIT(3380) RECFM(F B) LRECL(80)
                                +
                                                                00004200
                    BLKSIZE(6240) TRACKS SPACE(3 1) REUSE
/*                                                              00004300
                    */
                    OPENFILE SALIDA OUTPUT                      00004400
                    SET    &SALIDA = 'LA SALIDA CONTIENE:'      00004500
                    PUTFILE  SALIDA                             00004600
/*                                                              00004700
*/
                    SET &RC = 0                                 00004800
```

la línea :

5100 Carga en la variable RC el contenido del ultimo código de retorno.
Este paso es necesario ya que se machaca el contenido de la variable del sistema &LASTCC cuando se ejecuta una función, por ejemplo un WRITE.

5200 Examina el contenido del código de retorno que ha provocado la excepción. Si es 400 escribe en el terminal:
FIN NORMAL DE LA CLIST

5400 Si no es 400, ejecutara el conjunto de instrucciones enmarcadas entre DO y END.(Líneas 5500 a 6000)

5600 Escribe en el terminal mensajes para permitir la depuración de este

5700 Error anormal. Escribe código de retorno

5800 Escribe el contenido del buffer

5900 Termina la ejecución de la Clist.

El siguiente grupo de instrucciones se encarga de grabar en el fichero de salida el contenido de la variable indexada TABLA (nombre contenido en la variable NOMBRE, línea 1200)

```
ERROR +                                                    00004900
                                                           00005000
            DO
                                                           00005100
      SET &RC = &LASTCC
                                                           00005200
      IF &RC = 400 THEN +
                                                           00005300
WRITE FIN NORMAL DE LA CLIST
                                                           00005400
      ELSE +
                                                           00005500
DO
                                                           00005600
            WRITE INESPERADO FIN DE CLIST
                                                           00005700
      WRITE RC ES &RC
                                                           00005800
      WRITE LÍNEA
                                                           00005900
      EXIT
                                                           00006000
            END
                                                           00006100
END
```

La línea :

6300 Inicializa la variable CONTA con el valor 6 dado que las 6 primeras líneas contienen blancos y mensajes que no interesa grabar.

6400 Incrementa el contenido de la variable CONTA en 1.

```
/*    */                                                   00006200
      SET                                                  00006300
      &CONTA = 6
      SET                                                  00006400
      GRABA
R:    &CONTRA = &CONTRA + 1
      IF                                                   00006500
      &CONTRA > &ÚLTIMO THEN GOTO CERRAR
      SET                                                  00006600
      LINEA = &&NOMBRE&
      SET                                                  00006700
      &SALIDA = &LINEA
      PUTFILE  SALIDA                                      00006800
      /*WRITE                                              00006900
      &SALIDA
         */
      GOTO                                                 00007000
      GRABAR
      CLOSFILE SALIDA                                      00007100
CERRA
R:
      FREE FILE (SALIDA)                                   00007200
```

6500 En el caso de que el índice supere el valor del número de líneas capturadas, cede el control al párrafo CERRAR..

6600 Asigna a la variable temporal LINEA el contenido de la variable &NOMBREnnn

6700 Asigna al buffer asociado a la salida el contenido de la variable temporal LINEA.

6800 Graba en el disco el contenido del buffer SALIDA.

6900 Si no estuviera enmarcado como comentario permitiría escribir en el terminal el contenido del buffer escrito en el disco. Sirve para depuración.

7000 Cede el control incondicionalmente al párrafo GRABAR (Línea 6400).

7100 Cierra el fichero de salida

7200 Libera para el sistema el fichero asociado al nombre lógico de SALIDA.

Llegados a este punto, tenemos en el fichero &SYSUID..SALIDA la salida generada por la ejecución del comando LISTA de TSO que era el objetivo que nos habíamos marcado.

El siguiente paso podría consistir en presentar el contenido de un fichero en pantalla, en forma de lista a fin de poder manejar los nombres de las librerías que se han capturado con esta CLIST, pero ya partiendo de un fichero dado. Esto es lo que hace la clist que se ejecuta en la línea 8000 a la que se pasa como parámetro el nombre del fichero generado.

¿CÓMO PASAR UN FICHERO DESDE EL PC AL HOST?

En el disquete que se acompaña, se encuentra el fichero JMPALO00.ASC que contiene el listado del procedimiento comentado en formato ASCII, necesario para pasar la CLIST comentada al HOST.

Para poder pasar dicho fichero es necesario lo siguiente:

- **Primero:** que estemos trabajando desde un PC conectados al HOST

- **Segundo :** Tener sesión abierta en el HOST, y concretamente encontrarnos en el panel de ejecución de comandos de TSO.

- **Tercero:** Ejecutar desde el PC el comando :

```
/*                                                    00007300
 */
     /* ==== MUESTRA LA TABLA DE ELEMENTOS ===        00007400
=========                                  */
 /*                                                   00007500
 */
                                                      00008000
EXEC
 'JMPDES01.CLIST.INT(DISPLAY1)','&FICHERO'
```

SEND A:\JMPALO00.ASC 'dsn(miembro)' ASCII CRLF

Donde:

dsn es el nombre del dataset o fichero que se designa como receptor en el mainframe. En el caso de que sea una librería debe especificarse el nombre del miembro

```
Lista de librerías
ROW 1 TO 16 OF 94
OPCIÓN>
SCROLL ==> CSR

  Opciones : B Browse  <<< LIBRERÍAS ALO-
CADAS >>>_______
         E Edit
         M Miembros
          m                          Librerías
Observaciones
     - ------------------------------------------
_____ ------------
      DESJMP.ISPF.ISPPROF
      DESJMP.JMP.ESKELETO
      DESJMP.JMP.MENSAJES
      DESJMP.JMP.PANEL
      DESJMP.JMP.TABLAS
      DESJMP.SPFLOG1.LIST
      TERMFILE
****************************** BOTTOM OF DATA
```

Pantallas: Pantalla que muestra la ejecución del comando directamente desde línea de comando.

```
MENÚ PARTICULAR DE JMP
 OPTION  ===> tso lista

 1 Reorganización del GMP
 2 Tratamiento de tablas
 3 Tratamiento de cadenas
 4 miembros de una librería
 5 Borrar tabla
 6 Librerías alocadas
 7 Librerías especiales

 X EXIT      - SALIR DEL MENÚ

 TERMFILE
 DESJMP.JMP.PANEL
 DESJMP.ISPF.ISPPROF
 DESJMP.INT.TABLA
 ***
```

Pantalla que muestra mediante un panel la ejecución del mismo comando.

ASCII especifica que los datos se encuentran escritos en ese código, y no en el EBCDIC que es el que el usa normalmente IBM

CRLF : mantener el control de carro al final de la línea.

Nota: Este comando es el normal para la placa de comunicaciones IRMA, una de las más extendidas en el mercado, pero existen otros paquetes que hacen lo mismo aunque con un comando similar, por lo que se recomienda ejecutar desde el sistema del MS-DOS el comando **SEND /?** para conocer el formato correcto a emplear.

¿CÓMO EJECUTAR EL PROCEDIMIENTO COPIADO?

Para ejecutar esta clist en el HOST, al igual que cualquier otra clist, solo hay que escribir a nivel de comando :

TSO EX 'nombre-fichero-clist'

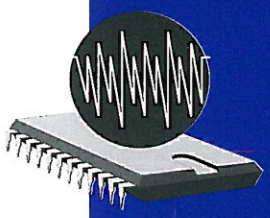
(Opción 6 del panel que sirve de ejemplo a la ejecución directa del comando). ■

```
/* <<<<<<<<<<<<<<<<<<<< JMPALO00 >>>>>>>>>>>>>>>>>>>>  00000100
*/                                                    00000100
 /* ============ CAPTURA Las librerías alocadas  =====*/ 00000200
PROC 0                                                00000300
   CONTROL MSG                                        00000400
   SET CABECERA = <<< Librerías alocadas >>>          00000500
   SET RC = 0                                         00000600
   CONTROL NOLIST NOCONLIST                           00000800
   SET &SYSOUTTRAP = 999                              00000900
   LISTA                                              00001000
   SET &SYSOUTTRAP = 0                                00001100
   SET &NOMBRE = TABLA                                00001200
   SET &CONTA  = 0                                    00001300
   SET &ULTIMO = &SYSOUTLINE                          00001400
/*              */                                    00001500

/* =============== MEMORIZAR SALIDA ===============
*/                                                    00001600
/*              */                                    00001700
LOOP1:  SET &CONTA  = &CONTA + 1                      00001800
   IF  &CONTA > &ULTIMO THEN GOTO LISTAR              00001900
   SET   &NOMBRE&CONTA = &&SYSOUTLINE&CONTA           00002000
   GOTO LOOP1                                         00002100
/*              */                                    00003000
LISTAR: SET &FICHERO= &SYSUID..JMP.CMD                00003100
/* WRITE        */                                    00003200
/* WRITENR LA SALIDA CONTIENE &ULTIMO LÍNEAS*/        00003300
/* WRITE        *                                     00003400
/*              */                                    00003500

/* ============ ESCRIBIR SALIDA EN FICHERO ==========
*/                                                    00003600
/*              */                                    00003700
   IF &SYSDSN('&FICHERO') = OK THEN +                 00003800
   ALLOC FILE(SALIDA) DATASET('&FICHERO') OLD REUSE   00003900
   ELSE ALLOC FILE(SALIDA) DATASET('&FICHERO') NEW CATALOG + 00004000
            UNIT(3380) RECFM(F B) LRECL(80) +         00004100
        BLKSIZE(6240) TRACKS SPACE(3 1) REUSE         00004200
/*              */                                    00004300
   OPENFILE SALIDA OUTPUT                             00004400
   SET          &SALIDA = 'LA SALIDA CONTIENE:'       00004500
   PUTFILE  SALIDA                                    00004600
/*              */                                    00004700
   SET &RC = 0                                        00004800
   ERROR +                                            00004900
   DO                                                 00005000
                SET &RC = &LASTCC                     00005100
                IF &RC = 400 THEN +                   00005200
                  WRITE FIN NORMAL DE LA CLIST        00005300
                ELSE  +                               00005400
                  DO                                  00005500
   WRITE INESPERADO FIN DE CLIST                      00005600
   WRITE RC ES &RC                                    00005700
   WRITE LINEA                                        00005800
   EXIT                                               00005900
                    END                               00006000
   END                                                00006100
/*              */                                    00006200
   SET &CONTA  = 6                                    00006300
GRABAR: SET  &CONTA  = &CONTA + 1                     00006400
   IF  &CONTA  > &ULTIMO THEN GOTO CERRAR             00006500
   SET &LINEA  = &&NOMBRE&CONTA                       00006600
   SET &SALIDA = &LINEA                               00006700
   PUTFILE  SALIDA                                    00006800
/* WRITE &SALIDA*/                                    00006900
   GOTO GRABAR                                        00007000
CERRAR: CLOSFILE SALIDA                               00007100
   FREE FILE(SALIDA)                                  00007200
/*              */                                    00007300

/* ======== MUESTRA LA TABLA DE ELEMENTOS ========
*/                                                    00007400
/*              */                                    00007500
 EXEC 'JMPDES01.CLIST.INT(DISPLAY1)','&FICHERO'       00008000
```

Listado general de la clist comentada.

EL TEMPORIZADOR 8253/8254

Enrique De Alarcón



Casi todos hemos necesitado en alguna ocasión realizar alguna animación que tenga movimiento constante, realizar reproducción de sonido digital por el altavoz, o calcular el tiempo que consume un determinado proceso, y todo ello de alguna forma que sea independiente de la velocidad del ordenador sobre el cuál se ejecuta. Como ejemplo de aplicaciones conocidas que usen el temporizador podríamos dar el nombre de DOOM y todos los arcade 3D de éste estilo de nueva aparición, juegos 3D que controla los movimientos reales de los personajes y objetos de forma constante e independiente-

El temporizador posee 3 contadores

mente de la velocidad de proceso de la CPU, no importando de este modo la cantidad de frames por segundo que sea capaz de realizar la aplicación y dando más realismo así al conjunto.

Lo cierto es que, de no haber usado éste sistema de control, la jugabilidad de éstos programas hubiese sido nula, ya que tanto iríamos "volando", como iríamos a paso de tortugas, dependiendo del tiempo que consumiese el cálculo de cada frame.

Si alguien todavía duda de la utilidad del temporizador, puede pensar, que sin éste, sería imposible tener la hora del reloj interno del DOS siempre actualizada. (podemos verlo escribiendo TIME desde la línea del DOS).

HARDWARE DEL SISTEMA

El reloj del sistema (8284A) funciona a una velocidad constante de 1193180 Hz, al cuál está conectado el temporizador (8253 timer o un 8254).

El temporizador posee tres contadores, cada uno de ellos con su propio <periodo> independiente, y los cuales son programables por nosotros. Cada vez que el reloj del sistema realiza un impulso, los tres contadores se incrementan en 1 y detectan si han llegado al periodo correspondiente para ponerse a 0 y realizar la acción correspondiente. Cada contador está dedicado a una tarea diferente:

El contador 1, genera una interrupción tipo 8 cada vez que llega al periodo programado. Esta interrupción apunta normalmente a una rutina del sistema operativo que realiza el control del tiempo del sistema. Cada vez que ésta interrupción se genera, la CPU deja la ejecución del programa en curso, y ejecuta la rutina asociada para después continuar con la ejecución donde la dejó.

El contador 2, se encarga de realizar el refresco de la memoria dinámica, activando el DMA cada vez que completa el periodo para actualizar la memoria. El tercer y último contador, está conectado al mismo tiempo al altavoz interno, el altavoz, y se encarga de generar un impulso de sonido cada vez que completa el periodo.

Como podrá deducir fácilmente, el primer y tercer contador pueden sernos útiles.

PROGRAMACIÓN DE LOS CONTADORES

Como hemos dicho, a cada contador

La posibilidad de poder controlar intervalos de tiempo reales con independencia del sistema en que corre una aplicación, se ha convertido en la base de funcionamiento de todas las aplicaciones más modernas.

se le puede programar un periodo distinto, por ejemplo, si queremos que el contador 1, trabaje a 11000 Hz, sólo tenemos que calcular el periodo que le corresponde en relación a la frecuencia de trabajo del reloj del sistema, y programarlo mediante los puertos correspondientes.

PERIODO = (1193180 / Frecuencia).

El contador 1, se programa mediante los puertos 43h y 40h y se realiza mediante los siguientes pasos:

Primero debemos enviar el valor 52 al puerto 43h para preparar al contador 1 del timer, para que reciba el periodo.

Después debemos enviar el periodo de 16bits en 2 bytes separados al puerto 40h, enviando primero el byte inferior y después el superior.

```
mov al,     52
mov dx,     43h
out dx,     al
mov dx,     40h
mov ax,     bx
out dx,     al
mov al,     ah
out dx,     al

El contador 3, se programa igual, pero usando los puertos 67 y 66:
mov al,     182
out 67,     al
mov ax,     bx
out 66,     al
mov al,     ah
out 66,     al
```

El código quedaría como sigue (dando de entrada el periodo en BX):

Debido a que el periodo es una palabra (16 bits) las frecuencias que se pueden generar con el temporizador van desde los 18.2 Hz (periodo de 65535) hasta un máximo de 1193180 Hz (periodo 0).

El temporizador también se puede activar y desactivar globalmente, ya que está controlado por un chip llamado PPI (8255 Programmable Peripheral Interface).

Para activarlo/desactivarlo sólo tenemos que poner a 1 o a 0, el bit 0

```
in al,      97
or al,      1
out 97,     al
```

del puerto B (97) del PPI. El código para activarlo quedaría como sigue:

EJEMPLOS DE APLICACIÓN

El caso más simple que se nos puede plantear, es el querer generar una sonido por el altavoz interno a una frecuencia X (o sea, activar/desactivar el altavoz X veces por segundo) sin necesidad de controlar cada activación/desactivación de sonido. Para ello, sólo necesitamos calcular el

```
in al,      97
or al,      2
out 97,     al
```

periodo al que corresponde la frecuencia que queremos generar, programarla en el contador 3 y activar el altavoz. Encargándose el temporizador de todo el trabajo por nosotros.

En éste caso, debemos poder saber activar/desactivar el altavoz, lo cuál se hace poniendo a 1 o a 0 el bit 1 del puerto B del PPI. El código para activarlo quedaría como sigue:

Un ejemplo más complejo que use las ventajas del primer contador, sería crear una rutina propia para el cálculo de tiempos de reloj con una precisión de centésimas, lo cual sería poder incrementar un variable 100 veces por segundo. Como el lector podrá deducir, nada más fácil que programar el periodo del contador 1 para que funcione a 100 Hz, y cambiar la interrupción 8 por una de usuario que incremente una variable cada vez que sea llamada, de ésta manera, obtendremos una variable que se incrementa 1 cada centésima de segundo y paralelamente al curso del programa principal en el que funcione, el cuál, podrá comprobar ésta variable siempre que la necesite.

Pasos a realizar:

1-Calcularemos el periodo que le daremos para que realice la interrupción 100 veces por segundo, que será 1193180/100, obteniendo 1193.

2-A continuación le daremos el periodo al contador 1 de la forma anteriormente descrita.

3-Cambiamos la rutina de servicio de la INT 8h por la nuestra (encargada de incrementar alguna variable que nos sirva de registro).

4-Activamos el temporizador.

Debemos controlar si la CPU puede soportar el trabajo que le demos a la INT 8h

Un tercer ejemplo común que use ambos contadores, podría ser reproducir un sonido digitalizado a 11 KHz mediante el altavoz interno.

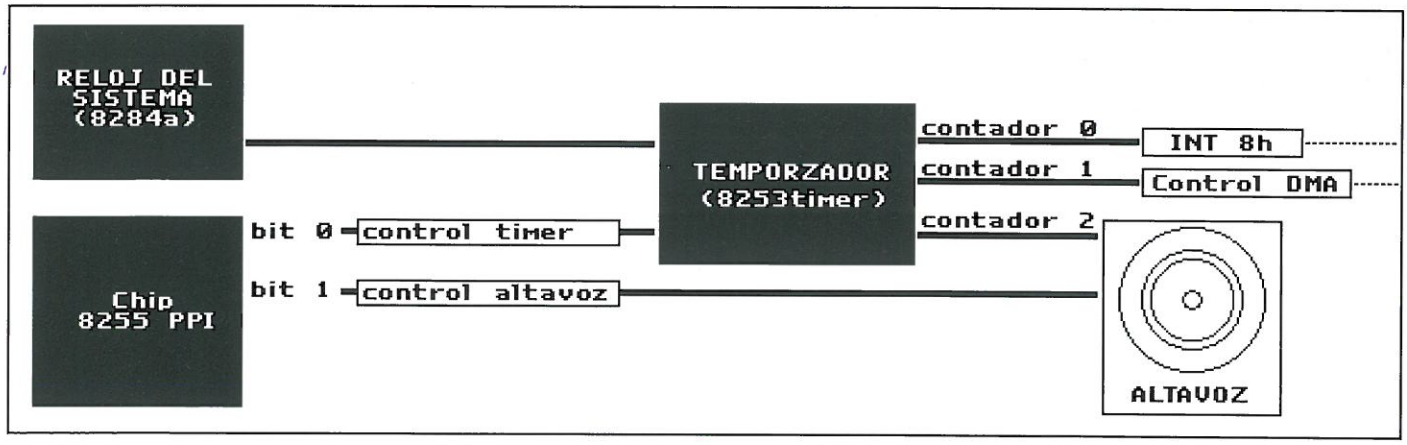
Por ello necesitamos una rutina que se active 11000 veces por segundo, y que vaya dejando al altavoz generando la última frecuencia dada hasta el siguiente Hz en el que daremos otro valor.

Como se puede deducir, tenemos que poner el contador 1 para que active la INT 8 once mil veces por segundo, y asociar a ésta una rutina propia que envíe cada vez al contador 3 un periodo correspondiente a la frecuencia del valor actual del sonido digital que se esté reproduciendo.

Mediante los bits 0 y 1 del chip PPI activamos/desactivamos el temporizador y el altavoz

Pasos a realizar:

1-Calcular el periodo que le daremos al contador 1 para que funcione a 11000 Hz, que será 1193180 /



Esquema de Funcionamiento del temporizador y sus conexiones.

11000, obteniendo un periodo de 108.

2-Programar el periodo obtenido en el contador 1.

Mediante los puertos 43h y 40h se accede al contador 1, los 67h y 66h acceden al contador 2

3-Cambiar la rutina asociada a la interrupción 8 por la encargada de enviar el sonido digital.

4-Activar el temporizador.

A partir de ahora, cuando el programa principal necesite generar un sonido, sólo será necesario poner en variables la dirección y duración del sonido y activar el altavoz, momento a partir del cuál, la rutina que ejecuta la INT 8 paralelamente a la aplicación se pondrá a enviar los valores de sonido automáticamente al altavoz.

ULTIMAS VALORACIONES

La forma mediante la cual se convierten los sonidos digitalizados de formato PCM (por ejemplo), que es el sistema que usa la SoundBlaster, a frecuencias de altavoz no se han explicado por que quedan fuera del tema.

Otro detalle a tener en cuenta es que, cuando ponemos el contador 1 a una frecuencia muy alta, debemos tener cuidado y calcular que la CPU pueda aguantar la carga de proceso de las interrupciones tipo 8. Por ejemplo, un 486 a 25 MHz, puede aguantar bien una rutina de gestión de sonido digital a 11000 Hz pero no un 386. En caso de no tenerlo en cuenta, podemos dejar al ordenador virtualmente parado o dejar al programa principal excesivamente lento.

El primer contador genera interrupciones tipo 8 y se programa mediante los puertos 43h y 40h

El control de tiempo real es fundamental en algunas aplicaciones

RESUMEN

- El temporizador está conectado al reloj del sistema.
- Posee 3 contadores programables por el usuario.
- El primer contador genera interrupciones tipo 8h y se programa mediante los puertos 43h y 40h.
- El segundo contador controla el DMA y el tercero controla el altavoz (speaker), y se programa mediante los puertos 67h y 66h.
- Los bits 0 y 1 del chip PPI (puerto 97) controlan el estado activo/inactivo del temporizador y del altavoz respectivamente.

Aplicaciones típicas son: Generación de sonidos de frecuencia X, calcular intervalos de tiempo o generación de sonido digital. ■

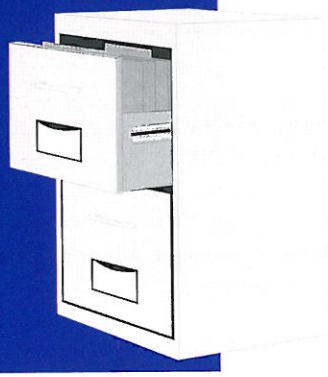
BIBLIOGRAFÍA

8088-8086/8087 Programación ENSAMBLADOR en entorno MS-DOS, Miguel Ángel Rodríguez Roselló. Anaya Multimedia.

8253 Programmable DMA Controller Data Sheet. Intel Microsystem Components Handbook.

PC interno 2.0. Programación del sistema. Michael Tischer. DATA BECKER, 1994.

Solucionario del programador de IBM PCs y compatibles. Robert Jourdain. Editorial ANAYA MULTIMEDIA.



NIVELES AVANZADOS DEL SISTEMA GET

Juan Manuel y Luis Martín

En el número anterior de esta revista se explicó la nueva forma de realizar operaciones de edición de variables y campos mediante la utilización de la programación orientada a objetos. Esta nueva forma esta basada en la utilización de objetos de la clase GET, conjuntamente con una serie de funciones especiales, que constituyen el denominado sistema get.

Como se mencionó en dicho artículo, existen tres posibles niveles de control sobre el proceso de edición. El primero, y más sencillo, es el nivel READ, que está basado en el propio mandato @...GET tradicional. En este artículo se explicarán los restantes niveles (nivel reader y nivel de funciones), que aunque presentan una mayor complejidad, permiten al programador un mayor control sobre el proceso de edición, posibilitando también una mayor personalización del mismo.

En el número anterior de esta revista se explicó la nueva forma de realizar operaciones de edición de variables y campos mediante la utilización de la programación orientada a objetos. Esta nueva forma está basada en la utilización de objetos de la clase GET, conjuntamente con una serie de funciones especiales, que constituyen el denominado sistema get.

EL NIVEL READER

Aunque la utilización del nivel READ permite realizar operaciones de edición bastante personalizadas, es posible que el programador se encuentre con situaciones en las que el control del proceso debe ser tan exhaustivo que no sea suficiente con alterar un poco el nivel READ. En este tipo de situaciones es necesario adentrarse en el siguiente nivel.

Como ya se mencionó, el nivel READ se encuentra implementado mediante la función ReadModal() del sistema Get, cuyo código fuente se encuentra disponible en el fichero GETSYS.PRG del directorio \CLIPPER5\SOURCE\SYS, y puede consultarse en el cuadro 1. Si se analiza su funcionamiento puede observarse que, tras una serie de operaciones de compatibilidad con versiones anteriores, su principal misión consiste en realizar el proceso de edición de cada uno de los objetos Get de la matriz GetList. Para ello, se realizarán sucesivas llamadas a una función, que es la encargada de realizar el proceso de edición propiamente dicho del objeto recibido como argumento. Este tipo de funciones se denominan *Reader*.

Las funciones *Reader* se encargan de editar los objetos recibidos como argumento

La utilización del nivel Reader está basada en la posible alteración o sustitución de la función Reader de cada objeto. Como puede observarse en el código de la función ReadModal(), Clipper utiliza por defecto una función denominada GetReader(), cuyo código fuente también puede ser consultado en el fichero GETSYS.PRG. Por tanto, la función GetReader() puede considerarse el reader por defecto.

Sin embargo, cuando el programador ha almacenado una función de usuario en el atributo reader del objeto, en forma de bloque de código, esta actuará como nueva función reader del objeto.

CUADRO 1

Código de la función *ReadModal()* estándar

```
FUNCTION ReadModal( GetList, nPos )

  LOCAL oGet
  LOCAL aSavGetSysVars

  IF ( VALTYPE( sbFormat ) == "B" )
    EVAL( sbFormat )
  ENDIF

  IF ( EMPTY( GetList ) )

    // Compatibilidad con Summer'87
    SETPOS( MAXROW() - 1, 0 )
    RETURN (.F.)              // NOTE

  ENDIF

  // Salvar las variables de estado
  aSavGetSysVars := ClearGetSysVars()

  // Almacenar los valores para la utilización de
SET KEY
  scReadProcName := PROCNAME( 1 )
  snReadProcLine := PROCLINE( 1 )

  // Seleccionar el primer GET ha editar
  IF !( VALTYPE( nPos ) == "N" .AND. nPos > 0 )
    nPos := Settle( Getlist, 0 )
  ENDIF

  WHILE !( nPos == 0 )

    // Obtener el siguiente GET de la lista y poner-
lo como GET activo
    PostActiveGet( oGet := GetList[ nPos ] )

    // Editar el GET
    IF ( VALTYPE( oGet:reader ) == "B" )
        EVAL( oGet:reader, oGet )    // Utilizar
el Reader del usuario
    ELSE
        GetReader( oGet )          // Utilizar el
Reader estándar
    ENDIF

    // Ir al siguiente GET en función de la condi-
ción de salida
    nPos := Settle( GetList, nPos )

  ENDDO

  // Restaurar las variables de estado
  RestoreGetSysVars( aSavGetSysVars )

  // Compatibilidad con Summer'87
  SETPOS( MAXROW() - 1, 0 )

  RETURN ( slUpdated )
```

EL READER POR DEFECTO

Antes de poder realizar el diseño de nuevas funciones reader, es conveniente conocer el funcionamiento del reader por defecto, es decir, de la función GetReader() del sistema Get. Esta función utiliza a su vez otras funciones del sistema Get que aún no han sido analizadas, y que pueden consultarse en la tabla 1.

La función GetPreValidate() realiza la evaluación de la cláusula de prevalidación del objeto Get (la cláusula WHEN del mandato @...GET). Para ello, se evalúa el bloque de código almacenado en el atributo PreBlock del objeto. En función del valor devuelto por el bloque de código, el atributo ExitState tomará los valores GE_WHEN (no se ha cumplido la condición) o GE_NOEXIT (se ha cumplido la condición).

La función GetPostValidate() realiza la evaluación de la cláusula de postvalidación del objeto Get (la cláusula VALID del mandato @...GET). Para ello, se evalúa el bloque de código almacenado en el atributo PostBlock del objeto. Si se devuelve el valor .T. (se ha cumplido la condición), el valor almacenado en el buffer de edición se asigna a la variable editada. Si por el contrario, se devuelve el valor .F. (no se ha cumplido la condición), el cursor se sitúa al principio del buffer de edición.

La función GetApplyKey() realiza las operaciones de edición apropiadas en función del valor de la tecla pasada como argumento. En esta función

CUADRO 2

Código de la función *GetReader()* estándar

```
PROCEDURE GetReader( oGet )

  // Editar si se satisface la condición WHEN
  IF ( GetPreValidate( oGet ) )

    // Activar el GET para editarlo
    oGet:setFocus()

    WHILE ( oGet:exitState == GE_NOEXIT )

          // Comprobar el TypeOut inicial (posi-
ciones no editables)
          IF ( oGet:typeOut )
            oGet:exitState := GE_ENTER
          ENDIF

          // Aplicar las teclas pulsadas hasta la
salida
          WHILE ( oGet:exitState == GE_NOE-
XIT )
            GetApplyKey( oGet, inkey( 0 ) )
          ENDDO

          // No permitir la salida si no se cumple
la condición VALID
          IF ( !GetPostValidate( oGet ) )
            oGet:exitState := GE_NOEXIT
          ENDIF
    ENDDO

    // Desactivar el GET
    oGet:killFocus()

  ENDIF

  RETURN
```

CUADRO 3

Función *MiReader()* para la edición con ayudas

```
FUNCTION MiReader( oGet )

  // Salvar el entorno y la pantalla
  LOCAL cPant := SAVESCREEN( 20, 16, 24, 65
),;
      cColor := SETCOLOR( "W/B+" )

  IF ( GetPreValidate( oGet ) )

    // Imprimir el texto de la ayuda
    @ 20, 16 CLEAR TO 24, 65
    @ 20, 16 TO 24, 65 DOUBLE
    @ 21, 17 SAY LEFT( oGet:Cargo, 48 ) COLOR
"BG/B+"
    @ 22, 17 SAY SUBSTR( oGet:Cargo, 49, 48 )
COLOR "BG/B+"
    @ 23, 17 SAY IF( SUBSTR( oGet:Cargo, 97, 48
) == "",;
      SPACE( 40 ), SUBSTR( oGet:Cargo, 97, 48 )
) COLOR "BG/B+"

    // Continuar la edición normal del GET
    oGet:setFocus()
    WHILE ( oGet:exitState == GE_NOEXIT )
      IF ( oGet:typeOut )
            oGet:exitState := GE_ENTER
          ENDIF
          WHILE ( oGet:exitState == GE_NOE-
XIT )
            GetApplyKey( oGet, inkey( 0 ) )
          ENDDO
          IF ( !GetPostValidate( oGet ) )
            oGet:exitState := GE_NOEXIT
          ENDIF
    ENDDO
    oGet:killFocus()

  ENDIF

  // Restaurar el entorno y la pantalla
  RESTSCREEN( 20, 16, 24, 65, cPant )
  SETCOLOR( cColor )

  RETURN NIL
```

se encuentran definidas todas las acciones posibles, dependiendo de la tecla que haya sido pulsada. Las teclas <Intro> y <Escape> actualizan el valor del atributo ExitState para permitir la salida.

Una vez analizadas estas funciones, es posible comprender con mayor facilidad el funcionamiento del reader por defecto, cuyo código puede consultarse en el cuadro 2. Como puede verse, el cuerpo de la función sólo se ejecuta cuando se cumple la cláusula de prevalidación. En ese caso, se asigna el foco al objeto GET, y se entra en un bucle de proceso hasta que la condición de salida no se cumpla. Después de verificar si se ha intentado sobrepasar el tamaño del buffer de edición, se realiza la lectura del teclado y las acciones apropiadas mediante la función GetApplyKey(), que además actualizará el valor del atributo ExitState. Para poder salir del bucle será necesario que se cumpla la cláusula de postvalidación. En ese caso, el objeto GET es desactivado, retirándole el foco.

CREACIÓN DE UN NUEVO READER

Para ilustrar las posibilidades de la utilización de un reader personalizado, se analizará un ejemplo en el que se muestra una ayuda referente al objeto que posee el foco. Para ello, será necesario almacenar el texto de dicha ayuda en el atributo cargo de cada uno de los objetos a editar. El cuadro 3 muestra el código del nuevo reader.

Como puede observarse, la función MiReader() salva el entorno y la zona de la pantalla que va a ser alterada. Si se cumple la cláusula de prevalidación, se dibuja una ventana en la pantalla con el texto de la ayuda. A continuación se asigna el foco al objeto, y se realizan las mismas operaciones que en el reader por defecto. Una vez que finaliza el proceso de edición, el objeto pierde el foco y se restaura el entorno y la zona de la pantalla alterada.

Cuando se utilizan este tipo de funciones, es posible aplicarlas a todos los objetos GET de la matriz GetList, o bien sólo a algunos, permitiendo que el resto utilice el reader por defecto. También es posible aplicar un reader

CUADRO 4

Código de la función *GetApplyKey()* estándar

```
PROCEDURE GetApplyKey( oGet, nKey )

  LOCAL cKey
  LOCAL bKeyBlock

  // Comprobar primero el valor de SET KEY
  IF !( ( bKeyBlock := setkey( nKey ) ) == NIL )
    GetDoSetKey( bKeyBlock, oGet )
    RETURN
  ENDIF
  DO CASE
  CASE ( nKey == K_UP )
    oGet:exitState := GE_UP

  CASE ( nKey == K_SH_TAB )
    oGet:exitState := GE_UP

  CASE ( nKey == K_DOWN )
    oGet:exitState := GE_DOWN

  CASE ( nKey == K_TAB )
    oGet:exitState := GE_DOWN

  CASE ( nKey == K_ENTER )
    oGet:exitState := GE_ENTER

  CASE ( nKey == K_ESC )
    IF ( SET( _SET_ESCAPE ) )

          oGet:undo()
          oGet:exitState := GE_ESCAPE

    ENDIF

  CASE ( nKey == K_PGUP )
    oGet:exitState := GE_WRITE

  CASE ( nKey == K_PGDN )
    oGet:exitState := GE_WRITE

  CASE ( nKey == K_CTRL_HOME )
    oGet:exitState := GE_TOP

#ifdef CTRL_END_SPECIAL

  // ^W y ^End van al último GET
  CASE ( nKey == K_CTRL_END )
    oGet:exitState := GE_BOTTOM

#else

  // ^W y ^End finalizan la edición (por defecto)
  CASE ( nKey == K_CTRL_W )
    oGet:exitState := GE_WRITE

#endif

  CASE ( nKey == K_INS )
    SET( _SET_INSERT, !SET( _SET_INSERT ) )
    ShowScoreboard()

  CASE ( nKey == K_UNDO )

    oGet:undo()
```

```
  CASE ( nKey == K_HOME )
    oGet:home()

  CASE ( nKey == K_END )
    oGet:end()

  CASE ( nKey == K_RIGHT )
    oGet:right()

  CASE ( nKey == K_LEFT )
    oGet:left()

  CASE ( nKey == K_CTRL_RIGHT )
    oGet:wordRight()

  CASE ( nKey == K_CTRL_LEFT )
    oGet:wordLeft()

  CASE ( nKey == K_BS )
    oGet:backSpace()

  CASE ( nKey == K_DEL )
    oGet:delete()

  CASE ( nKey == K_CTRL_T )
    oGet:delWordRight()

  CASE ( nKey == K_CTRL_Y )
    oGet:delEnd()

  CASE ( nKey == K_CTRL_BS )
    oGet:delWordLeft()

  OTHERWISE

    IF ( nKey >= 32 .AND. nKey <= 255 )

          cKey := CHR( nKey )

          IF ( oGet:type == "N" .AND. ( cKey ==
"." .OR. cKey == "," ) )
            oGet:toDecPos()
          ELSE

            IF ( SET( _SET_INSERT ) )
              oGet:insert( cKey )
            ELSE
              oGet:overstrike( cKey )
            ENDIF

            IF ( oGet:typeOut )
              IF ( SET( _SET_BELL ) )
                    ?? CHR(7)
              ENDIF

              IF ( !SET( _SET_CONFIRM ) )
                          oGet:exitState    :=
GE_ENTER
              ENDIF
            ENDIF

          ENDIF

    ENDIF

  ENDCASE
  RETURN
```

diferente para cada uno de los objetos.

Como ya se mencionó al describir el nivel superior, la función ReadModal() es la encargada de seleccionar el reader a utilizar para cada uno de los objetos. Si en el atributo reader del objeto ha sido almacenada una llamada a función (en forma de bloque de código), será esta la que se utilice para realizar la edición. En caso contrario, se utilizará la función GetReader() del sistema.

```
// Lee el Get
IF ( VALTYPE( oGet:Reader ) == "B" )
   EVAL( oGet:Reader, oGet )  // Usa un bloque reader propio
   ELSE
    GetReader( oGet )  // Usa el reader estándar
   ENDIF
```

Para que un objeto GET utilice como reader una función de usuario, será necesario almacenar ésta en su atributo reader. Para ello, puede utilizarse la cláusula SEND del propio mandato @...GET.

```
   @ 10, 10 GET ... SEND Reader := { | oGet | MiReader( oGet ) }
```

También es posible asignar directamente el bloque de código al atributo del objeto. Para ello, debe hacerse uso de la función ATAIL(), ya que ésta permite el acceso al último objeto GET creado, es decir, el último elemento de la matriz GetList. Además, de este modo es posible liberar la cláusula SEND para otros usos.

```
   @ 10, 10 GET ...
   ATAIL( GetList ):Reader := { | oGet | MiReader( oGet ) }
```

TABLA 1

Función	Descripción
ReadModal()	Función READ estándar para la edición de objetos Get
GetReader()	Función Reader por defecto de un objeto Get
GetApplyKey()	Función estándar para el procesamiento de teclas
GetPreValidate()	Evalúa la cláusula de prevalidación
GetPostValidate()	Evalúa la cláusula de postvalidación
GetDoSetKey()	Ejecuta un bloque de código de tipo SET KEY
ClearGetSysVars()	Salva y reinicializa las variables del sistema Get
RestoreGetSysVars()	Restaura las variables del sistema Get
GetActive()	Devuelve el objeto Get activo
ReadKill()	Fuerza la salida del objeto Get que se está editando
ReadUpDated()	Indica si se ha producido algún cambio en buffer

Principales funciones de soporte del sistema Get.

Para programadores en CA-Clipper

Five

Product líder

No lo dudes. Este producto es el que necesitas para hacer tu aplicación en Windows. Extremadamente fácil y potente. Resultados inmediatos. Más de 4.000 usuarios en todo el mundo.

FiveWin
Librería de CA-Clipper para Microsoft Windows
Realiza tus aplicaciones en Microsoft Windows con una presentación y funcionalidad totalmente profesionales. Elegido por la revista Americana Reference(Clipper) como el mejor producto para xBase en Windows (enero 94)
Precio promocional: 14.000 pts.

FiveOS2
Librería de CA-Clipper para IBM-OS2
Ahora puedes realizar aplicaciones de gestión para IBM-OS2 con la misma facilidad que para Windows. El entorno IBM-OS2 es el máximo competidor de Windows y se está haciendo muy popular. Aprovecha esta oportunidad.
Precio promocional: 14.000 pts.

FiveDos
Librería de CA-Clipper para MsDos
Un completo entorno de desarrollo para MsDos con ventanas, ratón, controles CUA y ejecución NoModal. Simula totalmente la funcionalidad de Windows y OS2.
Precio promocional: 14.000 pts.

¡ FiveWin en Vídeo !
Aprende ahora rápida y cómodamente la manera de realizar aplicaciones profesionales de gestión en Windows, tan fácilmente como ver una película de vídeo
Volumen 1.....8.000 pts.

Antonio Linares - Software
Urb. El Rosario, Avda. Rosario 34-A. 29600 Marbella - España
Tfno./fax: 95-2834830 BBS: 95-2213374 / 908 453368 voz

Distribuidor Nacional
Mail Simons S.L. Apdo. 2643. 28080 Madrid. Tfno. 91-5634486
BBS: 91-5637872 FAX: 91-5634451 E-mail: bmartinez @ simons.es

FiveWin

CA-Clipper es una marca registrada de Computer Associates, Windows es una marca registrada de Microsoft Corporation, OS2 es una marca registrada de IBM

DBU para Windows: Gestor de ficheros dBase para programadores. Soporta índices Clipper NTX, dBase III, dBase IV, Foxpro (Comix, Six 2.0) y NSX. PVP: 8.500 Ptas. FORMACIÓN, ASESORAMIENTO y CURSOS de programación WINDOWS con FiveWin en Madrid: ORTIZ DE ZÚÑIGA. Tfno: 91-575 20 47.

OTROS DISTRIBUIDORES

ARGENTINA
GO CLIPPER BBS&
FIVEWIN
Panamericana 3787
1609 Boulogne
ARGENTINA
+541-737-2767 Voz
+54-1-943-0563 Fax
+54-1-790-1906 BBS

BRASIL
VENEZUELA
CUBA
PARAGUAY
URUGUAY
SE BUSCA
DISTRIBUIDOR

CHILE
PROGRAMMER'S
HOUSE
Bustos 2157 dpto 13
Providencia/
Santiago
CHILE
+56-2-2322948 Voz
+56-2-2322948 Fax

CHILE
COMPUDATA
TUCAPEL 564 of. 77
CONCEPCIÓN
+56-41246035 Voz
+56-41542184 Fax

MÉXICO
PERU
COLOMBIA
BOLIVIA
SE BUSCA
DISTRIBUIDOR

PORTUGAL
MULTDIGITAL LDA
Pr. Manuel Guedes,
13 - 4º - Sala 17
+351-2-4647050 Voz
+351-2-4647051 Fax
+351-2-4647052 bbs
joaquím boavida @ tail. pt

La última versión la puedes encontrar en las siguientes revistas, CD-ROM's, BBS's o direcciones de INTERNERT: CD-ROM de PC MEDICA, Sólo Programadores, PC-ACTUAL, KENDER, AMS (Mail Simons s.l.) y las siguientes BBS's: bAuHaUs (+34-5-2213374), KENDER (944763506), CIBERLINEA (902-238624)., CIBERNETIX (986-691525) ftp. eunet. es: pub/InterStand/Simons.

El nivel más bajo del sistema Get es el nivel de funciones

EL NIVEL DE FUNCIONES

El nivel más bajo del sistema Get es el nivel de funciones. Consiste en realizar el proceso de edición aplicando las funciones del sistema Get y los métodos de la clase. Este nivel proporciona al programador un control total sobre el proceso de edición, ya que será él mismo quien decida todas las acciones que serán realizadas en función de la tecla que pulse el usuario.

Sin embargo, este nivel aumenta considerablemente la complejidad de los programas, lo que puede traer consigo un incremento en el número de errores. Por tanto, la utilización de este nivel no es recomendable salvo en casos excepcionales.

TABLA 2

Método	Descripción
ASSIGN()	Asigna a la variable editada el contenido del buffer
COLORDISP()	Asigna un nuevo color al objeto Get
DISPLAY()	Visualiza el Get en la pantalla
KILLFOCUS()	Retira el foco del objeto Get (lo desactiva)
RESET()	Reinicializa el Get con el valor inicial de la variable editada
SETFOCUS()	Asigna el foco al objeto Get, (lo activa)
UNDO()	Deshace los cambios realizados el buffer
UNTRANSFORM()	Convierte la cadena del buffer al tipo de datos editado
UPDATEBUFFER()	Rellena el buffer con el valor de la variable editada y vuelve a visualizar el Get
VARGET()	Devuelve el valor actual de la variable editada
VARPUT()	Asigna un valor a la variable editada
END()	Sitúa el cursor en el último carácter (a la derecha)
HOME()	Sitúa el cursor en el primer carácter (a la izquierda)
LEFT()	Desplaza el cursor un carácter a la izquierda
RIGHT()	Desplaza el cursor un carácter a la derecha
TODECPOS()	Sitúa el cursor en el carácter siguiente al punto decimal
WORDLEFT()	Desplaza el cursor una palabra a la izquierda
WORDRIGHT()	Desplaza el cursor una palabra a la derecha
BACKSPACE()	Borra el carácter anterior al del cursor
DELETE()	Borra el carácter sobre el que se encuentra el cursor
DELEND()	Borra todos los caracteres desde el cursor al final
DELLEFT()	Borra el carácter situado a la izquierda del cursor
DELRIGHT()	Borra el carácter situado a la derecha del cursor
DELWORDLEFT()	Borra la palabra situada a la izquierda del cursor
DELWORDRIGHT()	Borra la palabra situada a la derecha del cursor
INSERT()	Inserta un carácter en el buffer de edición
OVERSTRIKE()	Sobrescribe un carácter en el buffer de edición

Tabla 2. Métodos de la clase Get.

Para la realización del proceso de edición a nivel de funciones, es conveniente conocer tanto los métodos de la clase Get (véase tabla 2), como el resto de las funciones que proporcionan soporte al sistema Get (véase tabla 1).

El trabajo a nivel de funciones consiste básicamente en el rediseño de la función GetApplyKey() encargada de realizar el proceso de edición propiamente dicho. Como puede suponerse, la realización de una función totalmente nueva de éste tipo es una tarea bastante ardua, ya que será necesario describir todas las acciones a realizar para cada una de las teclas pulsadas. En el cuadro 4 se muestra el código de la función GetApplyKey() estándar. Como puede observarse, consiste en una gran estructura CASE con bifurcaciones para cada una de las posibles teclas.

Para utilizar una función GetApplyKey() nueva, será necesario alterar el nivel Reader, ya que es éste el encargado de llamarla. Para ello, pueden utilizarse dos métodos.

El primero consiste en alterar la función GetReader() del archivo GETSYS.PRG, cambiando la llamada a GetApplyKey() por una llamada a una función de usuario. Sin embargo, no es recomendable utilizar este método, ya que el archivo GETSYS.PRG quedará alterado para las sucesivas aplicaciones.

El segundo método consiste en la realización de un nuevo reader que contemple la llamada a la nueva función de proceso de teclas, tal y como se ha analizado en el apartado anterior.

CONCLUSIÓN

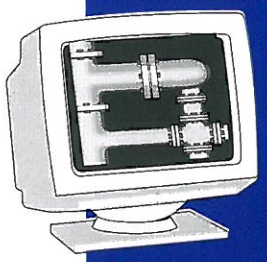
Como ha podido observarse a lo largo de los dos artículos dedicados al sistema Get, la inclusión de la Programación Orientada a Objetos en la edición de campos y variables proporciona al programador un control exhaustivo de dicho proceso.

El dominio de los tres niveles analizados permitirá al programador desarrollar aplicaciones con un interfaz de usuario mucho más profesional. Sin embargo, es recomendable utilizar el nivel más alto posible, siempre que éste se ajuste a las necesidades del programador. ■

La función ReadModal () es la encargada de seleccionar el *reader* a utilizar para cada uno de los objetos

WATCOM, EL COMPILADOR DE MODA

Agustín Guillén



Su mayor éxito ha sido "socializar" las aplicaciones de 32 bits bajo MS-DOS, esto es, trabajar en modo protegido sin interferir apenas en el desarrollo de un programa; pero aumentando considerablemente las posibilidades del DOS y rompiendo la barrera de los 640 Kb.

Se trata de un paquete muy completo que ocupa, si se instala completo, unos 200 Mb en el disco duro. En el CDROM incluido en el paquete distribuido, se encuentra tanto la versión para instalar, como la instalada, lista para ejecutar con sólo preparar unas variables de entorno. Casi todas las herramientas y utilidades vienen en versiones para MS-DOS (menos el IDE), OS/2, Windows y Windows NT.

Se trata de un paquete muy completo que ocupa, si se instala completo, unos 200 Mb en el disco duro

Las posibilidades de generación de ejecutables o aplicaciones a crear son casi ilimitadas. Se pueden producir ficheros de los siguientes tipos:

Windows 32-bits
- Ejecutables
- Librerías dinámicas (DLL)
- Librerías (LIB)
- Ficheros de ayuda (HLP)

Windows 16-bits
- Ejecutables
- Librerías dinámicas (DLL)
- Librerías (LIB)
- Ficheros de ayuda (HLP)
- Ejecutables MFC 2.5
- Librerías dinámicas MFC 2.5

DOS 16 bits
- Ejecutables (EXE y COM)
- Librerías (LIB)

DOS 32 bits
- Ejecutables DOS4GW
- Ejecutables Pharlap (EXP)
- Ejecutables Pharlap TNT (EXE)
- Ejecutables Flashtek (EXE)
- Ejecutables Flashtek basados en la pila (EXE)
- Librerías (LIB)

Novell Netware 32 bits
- Módulos NLM

Autocad 32 bits
- Ejecutables (EXP)
- Controladores de dispositivos (EXP)
- Controladores de dispositivos de coma flotante (EXP)

Windows NT 32 bits
- Ejecutable en modo carácter (EXE)
- Ejecutable en ventana (EXE)
- Librería dinámica (DLL)
- Librería (LIB)
- Fichero de ayuda (HLP)
- Librería dinámica MFC 2.1 (DLL)
- Ejecutable en ventana MFC 2.1 (EXE)

OS/2 32 bits
- Ejecutable (EXE)

Si existe un compilador o paquete de desarrollo del que últimamente nunca se deja de hablar, que vemos aparecer como cabecera de los mejores juegos del momento o como referencia en revistas especializadas cuando de prestaciones se trata, ese es el Watcom/386 C++. Ahora con su reciente versión, la 10.0a, se actualiza, completa y revaloriza todavía más si cabe.

Ejecutable PM (EXE)
Ejecutable en pantalla completa (EXE)
Dispositivo Virtual (EXE)
Librería dinámica (DLL)

Watcom 10.0a incorpora editores basados en el clásico y popular vi de Unix

Librería (LIB)
Fichero de ayuda (HLP)

OS/2 16 bits
Ejecutable (EXE)
Dispositivo Físico (EXE)
Librería dinámica (DLL)
Librería (LIB)
Fichero de ayuda (HLP)

Además, incorpora las siguientes librerías o paquetes de desarrollo:

- OS/2 2.x Toolkit
- Soporte para WorkFrame/2
- Soporte para las MFC (Microsoft Foundation Classes) versión 2.1 de 32 bits y para la versión 2.5 de 16 bits.
- Librerías para desarrollar en Microsoft Win32s (ejecutables de 32 bits que corren bajo Windows 3.x de 16 bits).
- Código fuente de la runtime de arranque.
- SOMobjects Toolkit para OS/2 2.x.
- Soporte de desarrollo para Novell Netware.

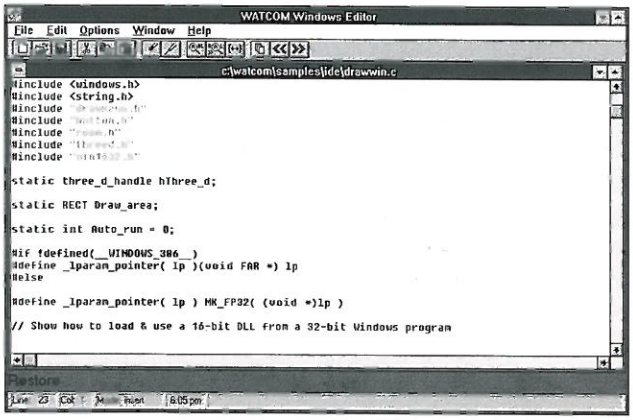
La información de ayuda on-line viene duplicada para cada plataforma de desarrollo, es bastante completa y suele abarcar los siguientes temas:

- Descripción de las librerías de C y C++.
- Información general sobre el compilador Watcom.
- Ayuda sobre el enlazador (linker) y el depurador (debugger).
- Ayuda sobre las diferentes herramientas auxiliares del paquete.
- Lista del API completo del Windows (16 y 32 bits).
- Descripción de las MFC.

IDE

El IDE (Integrated Development Environment) o entorno integrado de desarrollo, es la aplicación que se encarga de enlazar a las otras herramientas de manera fácil e intuitiva, permitiendo el manejo de grandes proyectos sin necesidad de la creación de enormes ficheros make. Permite también llamar a cada una de las herramientas auxiliares existentes en el paquete.

El entorno muestra todos los elementos relacionados con un proyecto y permite trabajar con partes del mismo, creando librerías o subproyectos antes de su unión para la obtención del programa final. Su principal función es la de configurar rápidamente los parámetros del compilador, enlazador, depurador, herramienta make o del compilador de recursos; gracias a las múltiples y ordenadas pantallas de switches a las cuales se tiene acceso.



Editor de fuentes.

Una función muy interesante es la posibilidad de visualizar la línea de comandos necesaria para crear el ejecutable del proyecto activo, o para compilar el fichero fuente seleccionado. De ese modo se pueden comparar las opciones cambiadas en las cajas de diálogo del IDE con su parámetro correspondiente en el programa utilizado.

La función del IDE es la de facilitar el manejo de los proyectos, pero no es imprescindible de cara a la consecución de un ejecutable (cuando se desarrolla bajo MS-DOS, no se dispone de ningún entorno IDE).

EDITOR

Watcom 10.0a incorpora varios editores, todos ellos basados en el clásico y popular vi de Unix. Seguro que esto generará amores u odios a más de uno. Las versiones que vienen incluidas son: Windows, Windows NT, OS/2 y MS-DOS; todas ellas gráficas menos la de MS-DOS, siendo este último una réplica de la versión bajo Unix (eso sí, con la posibilidad de usar un menú superior de opciones). Las versiones bajo sistemas operativos gráficos son sencillas pero funcionales. Permiten cambiar fácilmente el tipo de letra, su tamaño y, mediante una paleta de colores, los colores del syntax highlighting (remarcación mediante colores de las distintas partes de un fuente: palabras reservadas, comentarios, comandos del preprocesador, etc). Disponen, asimismo, de una práctica barra de herramientas, que contiene las operaciones más usuales en la edición de un fichero.

Como características destacables están la posibilidad de ajustar el syntax highlighting para trabajar con ficheros fuente de C, C++ o Fortran y la línea de estado inferior que muestra en grandes letras la última acción realizada o la posición en la que nos encontramos dentro del fichero editado.

COMPILADORES

Los compiladores vienen en versión 286 (16 bits) y en versión 386 (32 bits). También se dispone de versiones para C y para C++, quedando de la siguiente forma:

WCC.EXE	C, desarrollos en 16 bits.
WCC386.EXE	C, desarrollos en 32 bits.
WPP.EXE	C++, desarrollos en 16 bits.

WPP386.EXE C++, desarrollos en 32 bits.

Uno de los platos fuertes del paquete Watcom son sus numerosas optimizaciones, pues permite manipular sobre 20 diferentes parámetros para obtener justo lo deseado.

ENLAZADOR (LINKER)

Sorprendentemente sólo viene un programa para enlazar (en tres versiones ejecutables diferentes: DOS/Windows, Windows NT y OS/2), que se encarga de crear casi cualquier tipo de ejecutable o librería imaginable. Son tantas las opciones posibles, que han tenido que agruparlas según la plataforma destino en la que correrá el ejecutable a crear. Aún así se hace imprescindible utilizar un fichero de directivas (.LNK) el que especificar todas las opciones deseadas. Para una información detallada de los parámetros y directivas posibles basta teclear "WLINK /?".

DEPURADOR (DEBUG)

Existen versiones en modo texto (WD) y en modo gráfico (WDW), para cada una de las plataformas posibles (DOS, WINDOWS, NT y OS/2). Watcom ha mejorado el interfaz antiguo de sus depuradores, famosos por lo "arduo" que era trabajar con ellos, y ahora ofrece uno basado en ventanas y ratón, permitiendo además el modo antiguo a base de comandos, cuya potencia era incuestionable.

Watcom ha utilizado toda la potencia de los 386 y ofrece incluso depuración hacia atrás o Undo, disponible hasta hace poco sólo en potentes y caros emuladores cruzados de procesadores.

HERRAMIENTAS AUXILIARES

Watcom viene con infinidad de herramientas o aplicaciones auxiliares para la programación. Se trata de utilidades de muy diversos tipos, desde información del sistema en el que se ejecutan, hasta programas de otros fabricantes, a los que han sido comprados sus derechos. Algunas de las herramientas más prácticas o interesantes son:

- Los servidores para depurar en un ordenador remoto, ya sea por conexión paralela, serie o cualquier tipo de red.
- Rminfo y Pminfo, que muestran información sobre el sistema en modo protegido o en modo real.
- Extensores para ejecutar aplicaciones MS-DOS de 32 bits.
- Visualizadores de ficheros de ayuda (WHELP).

Watcom ha mejorado el interfaz antiguo de sus depuradores

- Compactador de la información de depuración (CVPACK).
- Gestor de librerías (WLIB)
- Compilador de recursos (WRC)
- Ensamblador (WASM)
- Desensamblador de OBJ (WDISASM).
- Compilador de ficheros help (HC30 y HC31).
- Utilidad para compilar y enlazar de una sola vez (WCL y WCL386).
- Editor de cajas de diálogo (WDE).
- Editor de recursos (WRE).
- Herramientas para el control de los mensajes y de la pila del sistema (WSPY, WHEAPWLK, WDDESPY)

PRESTACIONES

Para comprobar una de las características más importantes de un compilador, la velocidad de los programas creados, se han realizado unos pequeños programas de prueba. El contrincante ha sido el famoso Borland C++ 4.0, y en ambos se han activado las máximas optimizaciones posibles para mejorar la velocidad final. Las pruebas se han realizado bajo un 486 DX a 25 Mhz, y se han creado ejecutables, tanto de 16 bits como de 32 bits en el caso de Watcom.

Las pruebas realizadas han sido:

- Cálculo de operaciones con coma flotante, utilizando senos, cosenos, tangentes y operaciones matemáticas clásicas:
  Bucles realizados: 100.000
  Borland: 12 sg.
  Watcom 16: 8 sg.
  Watcom 32: 6 sg.
- Manejo de cadenas mediante punteros, asignaciones y comparaciones:
  Bucles realizados: 1.000.000
  Borland: 38 sg.
  Watcom 16: 33 sg.
  Watcom 32: 31 sg.
- Cálculo convencional intensivo con números de 16 bits:
  Bucles realizados: 1.000.000
  Borland: 3 sg.
  Watcom 16: 3 sg.
  Watcom 32: 5 sg.
- Cálculo convencional intensivo con números de 32 bits:
  Bucles realizados: 1.000.000
  Borland: 40 sg.
  Watcom 16: 8 sg.
  Watcom 32: 4 sg.
- Salida a pantalla mediante la función printf:
  Bucles realizados: 1.000
  Borland: 21 sg.
  Watcom 16: 21 sg.
  Watcom 32: 20 sg.

Como se puede apreciar, la potencia superior de los ejecutables de Watcom 32 bits en casi cualquier tipo de operación (lógicamente exceptuando el manejo de números de sólo 16 bits) es evidente; incluso en áreas que tradicionalmente eran muy lentas cuando se utilizaban extensores para el MS/DOS, tales como la salida a pantalla.

Parece ser que el intercambio entre modo real y modo protegido se ha acelerado hasta el punto de hacerlo casi transparente.

RESUMEN

Nos encontramos ante uno de los paquetes de desarrollo más completos y potentes del mercado, en el que, gracias a su diversidad, tenemos ampliamente asegurada la inversión. Sólo se podría pedir un entorno integrado de desarrollo para MS-DOS, pero como al parecer la próxima versión de Windows, va a "acabar" con él, quizá no se eche en falta en un futuro próximo. ■

CORREO DEL LECTOR

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relaccionado con la programación.

Estoy utilizando el compilador de C++ que regalaron en un número anterior, pero me falta información sobre las librerías que incorpora. ¿Tienen algo que ver con las de Borland o Microsoft? Gracias y adelante con tan práctica revista.

Manuel Sergio Jaime Rodríguez
(Málaga)

Desde luego las librerías estándar son las mismas (o casi) que en Borland o Microsoft y si dispone de información de otros compiladores, le valdrá igualmente. Pero, para el resto de las funciones, y para una mayor comodidad, puede utilizar la ayuda on-line que incorpora Symantec, la cual está muy bien estructurada y le permitirá navegar y buscar incluso mejor que en un manual de referencia.

Llevo un tiempo intentando adaptar el programa de visualización de ficheros gráficos PCX, que trabaja exclusivamente con ficheros 320x200 y 256 colores, para que me muestre imágenes de 640x480 o 1024x768. Por más que "trasteo", no lo consigo y me muestra imágenes rotas de todo tipo. Ayuda, gracias.

Luis Carlos Gómez Rodríguez
(Piedras Blancas - Asturias)

La modificación es sencilla, solo hay que cambiar los controles y la gestión del ancho y el alto de pantalla. La parte más complicada es la referente al acceso a la memoria de vídeo. En el modo 320x200, se ocupan 64.000 bytes de memoria de vídeo, pero en el modo 1024x768 son 768.432 bytes. Como en el PC solo se dispone de 64K (un segmento) para la memoria de vídeo, basta con un segmento para el primero, pero para el segundo, hay que recurrir a la paginación o segmentación. Este procedimiento no es estándar (al igual que cualquiera de los modos gráficos que lo necesiten) y cada VGA (mejor dicho, SVGA) tiene su propia forma para realizarlo. Para acabar con todo este lío, se inventó la norma VESA, que mediante una BIOS que la contemplara (la mayoría de las tarjetas modernas) o mediante un driver externo, unificara el cambio de segmento y otras tareas dependientes de cada hardware. Este sistema es el más cómodo para trabajar con los modos gráficos de SVGA y es preferible que se adjunte con cada programa el driver VESA correspondientes (existe un driver común para todas las tarjetas, denominado UNIVESA), que andar pensando que tipo de hardware tiene instalado el ordenador. Volviendo a la pregunta, cuando se va volcando a la videoram el gráfico en cuestión, es necesario ir controlando si se alcanza el final de un segmento y si esto ocurre, cambiar al siguiente. Si no se realiza dicho proceso, se obtendrá el clásico efecto de ir redibujando una y otra vez porciones del gráfico sobre la misma zona de pantalla. En el disco que se adjunta con la revista, se incluye una descripción de la norma VESA y de sus funciones estándar.

Hace poco, al instalar la versión del Linux que entregaban con la revista, particioné mi disco duro mediante la utilidad FIPS, para no perder mis datos de MS-DOS. Pero desde entonces he notado un descenso en la velocidad de acceso a disco, hasta el punto de hacerse insoportable el trabajar en Windows. ¿Saben a qué se puede deber?, ¿influye el tamaño de las particiones en la velocidad de acceso de un disco duro? Gracias.

Juan Alfonso Pastor Sierra
(Cáceres)

Todo influye, pero no de manera apreciable. La única idea que se nos ocurre sobre el problema, es que al disminuir el tamaño de la parte MS-DOS, el Windows se haya quedado sin espacio para su fichero de intercambio, disminuyendo notablemente sus prestaciones.

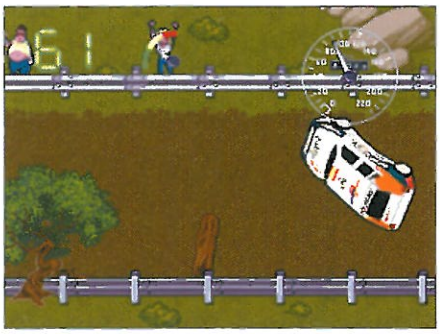
Me gustaría conectar dos ordenadores por el puerto paralelo y, usando el comando interlink del MS-DOS, comunicarlos. Ignoro cuales son las conexiones del cable pin a pin. ¿Podrían publicar el esquema del cableado? Un saludo.

Ricardo García Alcázar
(Villafranca del Castillo - Madrid)

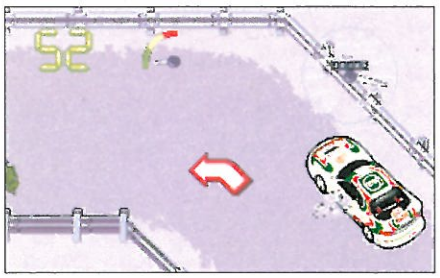
Buena idea, a menudo se recurre a costosos y complicados programas para tareas sencillas, en las que sirve incluso nuestro querido y odiado MS-DOS. En el disco que acompaña a la revista, dentro del subdirectorio CORREO, se incluye el archivo CABLE. TXT. En el se almacena el esquema para una conexión mediante el cable paralelo y el de una conexión utilizando los puertos serie.

TODA LA EMOCION DEL MUNDIAL DE RALLYS EN TU PC

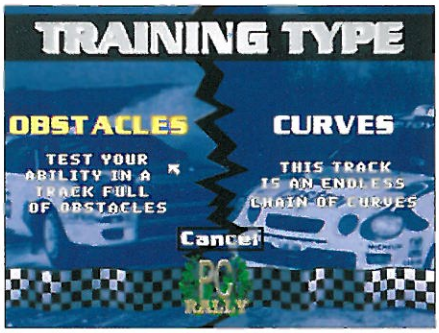
Evita los obstáculos con la ayuda de las instrucciones de tu copiloto.

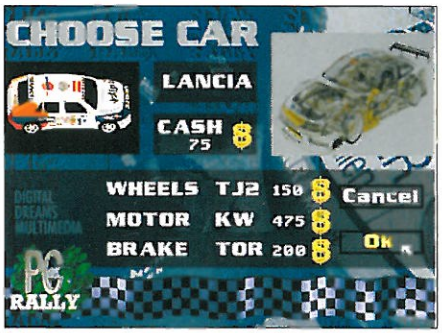


Gráficos y animaciones digitalizadas con sonido y música simultáneos.



30 Tramos diferentes de nieve, tierra y asfalto para poner a prueba tus reflejos.





¡YA A LA VENTA EN TU QUIOSCO!
POR SÓLO 2495 PTAS

CON LA GARANTÍA DE DDM DIGITAL DREAMS MULTIMEDIA

cita PC RALLY enviando este cupón o llamando al teléfono (91) 741.26.62 de 9 a 14 y de 15:30 a 18:30 o por Fax al (91) 3.20.60.72

eo que me envíen: ☐ **PC RALLY por 2495 ptas, + 250 ptas. de gastos de envío.** ☐

ombre y apellidos.....Domicilio.....Población.....

ovincia.....C.P .....Fecha de nacimiento.....Profesión.....

FORMA DE PAGO:

☐ Talón a TOWER COMMUNICATIONS SRL ☐ Contra-reembolso

☐ Giro Postal nº.....de fecha.....

Tarjeta de crédito VISA nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐

Teléfono .....

Firma ,

TOWER COMMUNICATIONS

Fecha de caducidad de la tarjeta.....Nombre del titular, si es distinto.....

Rellena este cupón y envíalo a: TOWER COMMUNICATIONS SRL C/ Marques de Portugalete 10, Bajo 28027 Madrid.

DESDE SIEMPRE TODOS HEMOS DESEADO MIRAR A TRAVÉS DE WINDOWS

AHORA YA PUEDE ABRIR SUS PROPIAS WINDOWS

ahora bien, WINDOWSI!

...PERO CON APELLIDOS :

Window Base 2.0

EL SISTEMA DE GESTIÓN DE BASES DE DATOS RELACIONAL BAJO WINDOWS

© SEDYCO, S.A.

SEDYCO, S.A. - Distribuidor Mayorista Exclusivo para España de SPI, Inc.
Francos Rodríguez, 64 - 7° B Esc. Izq. • 28039 Madrid • Tel. (91) 311 46 45 • Fax (91) 450 04 24 • E-Mail: sedyco@ibm.net